Automation of Operation and Testing for European Space Agency's OPS-SAT Mission

Felix Hessinger

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.
Espoo 12.08.2019

Thesis Supervisor

Prof. Esa Kallio

Thesis Advisor

David Evans

Aalto University
School of Electrical Engineering

Funded by the Erasmus+ Programme of the European Union

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Felix Hessinger

Title Automation of Operation and Testing for European Space Agency's OPS-SAT Mission

Degree programme Erasmus+ SpaceMaster

Major Space Robotics and Automation **Code of major** ELEC3047

Supervisor Prof. Esa Kallio

Advisor David Evans

Date 12.08.2019 **Number of pages** 130 **Language** English

Abstract

This thesis presents a solution for mission operation automation in European Space Agency's (ESA) OPS-SAT mission. To achieve this, the ESA internal mission automation system (MATIS) in combination with the mission control software (SCOS) are used. They control the satellite and all ground peripherals and programmes to enable fully automated and unsupervised satellite passes. The goal of this work is the transition from the existing manual operation, with a human operator watching over and controlling all systems, to an automated system. This system supports the operation engineer and replaces the operator himself. A large section of this thesis consists of the setup, configuration, integration of all programmes and virtual machines and testing of the MATIS software, as well as the Service Management Framework (SMF) which connects MATIS to non-MATIS applications like SCOS. During testing, many problems could be identified, not only OPS-SAT specific ones, but also general problems applying to all missions that consider using MATIS for future operation automation. These findings and bugs discovered during testing are reported to the responsible authorities and presented in this work. Further features of this thesis are the elaborations of the mission operation automation concept and the satellite pass concept, providing an in-depth view of the automation and passes of OPS-SAT as well as the general concepts and thoughts, which can be used by other missions to accelerate integration. An additional key feature of this thesis is the newly developed standard for operation notation in Excel, which has been achieved in close cooperation with the operation engineer. Furthermore, to accelerate the process of switching from manual to automated procedures, several converters have been developed iteratively with the new standard. These converters allow fast transformation from Excel to the procedure programming language called PLUTO used by MATIS.

Not only do the results and converters of this work accelerate the procedure integration by 80%, they also deliver a more stable mission automation system that can be used by other missions as well. Operation automation reduces the operational costs for satellites and space missions significantly, as well as reducing the human error to a minimum. Therefore, this thesis is the first step towards a future with complete automation in the area of satellite operations. Without this automation, future satellite cluster configurations, like Starlink from SpaceX, will not be possible to put into practice, due to their high complexity, exceeding the comprehensibility and reaction time of humans.

Keywords Automation, Converter, ESA, ESOC, European Space Agency, Innovation, MATIS, Mission, Mission Automation System, Operation, Procedure, OPS-SAT, PLUTO, SCOS, SMF, Space, Testing

"In any given moment, we have two options:
To step forward into growth, or to step back into safety."
-Abraham Maslow

Acknowledgement

I would like to thank the OPS-SAT team and close colleagues at ESA for their support, the great times made possible by the team and the friendships that evolved out of it. Without the enormous effort, communication, efficiency and harmony in the team, as well as overtime hours, this project would not have been possible. Furthermore, I am especially grateful for David Evans great work and his motivational and supportive character and for Dr. Daniela Taubert's outstanding assistance and great ideas during this project. Additionally, I would like to thank my girlfriend for her patience and support during this thesis. Finally, I give my appreciation to Professor Esa Kallio's competence and fast and supportive replies that sped up the writing of the thesis. It has been a pleasure to work with him.

I wish the OPS-SAT team all the best for the launch later this year and a successful and interesting journey during the commissioning phase of OPS-SAT and beyond.

Contents

Abbreviations

ACU	Array Conditioning Unit
APID	Application Identifier
AS	Automation System
AST	Arctic Space Technologies
AU	Application Unit
cADCS	Coars Attitude Determination and Control System
CCSDS	Consultative Committee for Space Data Systems
CFDP	CCSDS File Delivery Protocol
EGSE	Electrical Ground Support Equipment
EIRP	Equivalent Isotropically Radiated Power
EM	Engineering Model
ESA	European Space Agency
ESEC	European Space Security and Education Centre
ESOC	European Space Operations Centre
FARC	File Archive
FDIR	Fault Detection, Isolation and Recovery
FM	Flight Model
FMS	File Management Services
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
iADCS	Fine (Intelligent) Attitude Determination and Control System
ICD	Interface Control Document
Kbps	Kilo Bits per Second
LEOP	Launch and early Orbit Phase
MAUS	MATIS User Schedule
MAES	MATIS Event Schedule
MAPS	MATIS Planned Schedule
MATIS	Mission Automation System
Mbps	Mega Bits per Second
MIB	Mission Information Base
NIS	Network Interface System
OBC	On-Board Computer
OBSW	On-Board Software
OoL	Out of Limits
PLUTO	Procedure Language for Users in Test and Operations
POD	Picosatellite Orbital Deployer

RAM	Random-Access Memory
SCTL	System Control Display
SCOS	Spacecraft Control and Operation System
SDR	Software Defined Radio
SE	System Element
SEPP	Spacecraft Experimental Processing Platform
SLE	Space Link Extension
SLES	SUSE Linux Enterprise Server
SMF	Service Management Framework
SMILE	Small Mission Infrastructure Laboratory Environment
SPR	Software Problem Report
TC	Telecommand
TKMA	Task Management
TLE	Two Line Element
TM	Telemetry
TUG	Technical University Graz
UHF	Ultra High Frequency
VHF	Very High Frequency
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Motivation

Since the beginning of space missions, spacecraft operations have been a key element of mission success. For the public, a successful mission is often measured by the lifetime or scientific value of the spacecraft itself as well as by its measurements and discoveries [1]. Therefore, many people tend to forget about the ground operations running in the background at the control centres. Nevertheless, without the ground stations and their operations, all scientific data never would have reached the ground and therefore, would be useless. Additionally, without watching over the telemetry, reacting to it and sending new telecommands to the spacecraft, many missions would have failed a few days after the launch due to unforeseen scenarios. A very old but popular example for this is the Apollo 13 mission, where mission control saved the spacecraft and, most importantly, the crew [2]. Nevertheless, this is one of the few exceptions when mission control gained public credit for its work. With the increasing number of satellites, ranging from commercial industry to small satellites like cubesats, the space industry needs operation automation to keep up with the complexity and quantity of mission operations. In the near future this challenge will have to be tackled. Especially for large configurations like SpaceX's Starlink, OneWeb and Boeing's constellation, automation and new ground station capabilities are a necessity [3, 4]. Currently, these operations are often handled by highly paid operation engineers watching over the satellites at any time up to 24 hours a day. Therefore, ground operations cause significant costs during long-term space missions and satellite operations. Automating the ground operations will result in a great reduction of costs during operation. This cost reduction can enable smaller missions without a lot of budget to operate their spacecraft. This reduction of cost is a necessity for the OPS-SAT mission due to their technology demonstration goals and limited budget. Therefore, this Master's thesis is focusing on this topic, providing the foundation for OPS-SAT mission's operation automation.

1.2 Structure of Thesis

This thesis is divided into seven chapters. In the first chapter, a short introduction in the motivation and the state-of-the-art technology is provided and the concept behind cubesats and the OPS-SAT mission itself will be approached and explained. After this, an explanation of the main facilities and soft- and hardware components used during this project is given. In chapter three, the preparation of all systems is described and wrong or missing configuration is pointed out. These preparations mainly correspond with configurations and adaptations in the mission automation system (MATIS) and the mission control software (SCOS). Chapter four consists of the implementation of the operation automation and its integration from manual execution of procedures to MATIS. This integration introduces a newly developed standard for procedure writing and converters developed for OPS-SAT and potentially other missions. This chapter will also provide an insight into the most important functions used by the converter and visual examples of its conversion. Afterwards, chapter five delivers insights into the elaborated automation concept, its execution and the resulting information flow. Testing the created system with all its interconnected and interacting peripherals is realised in chapter six. Additionally, it provides a list of bugs that have been discovered during the work on this thesis and which have been reported to the corresponding support team or solved during this work. To finalise this thesis project, chapter seven concludes the findings, achievements, pros and cons of the used systems and presents an overall conclusion. Furthermore, this chapter catches a glimpse of the future of mission operation automation as well as a general trend in the space industry.

1.3 State of The Art

The current state of the art technology in automation of ground operations can be found in the PROBA missions [5, 6, 7, 8]. The PROBA team managed to automate their nominal passes and operations.
Their approach for automation is based on two main components: The mission planning tool, which is based on the orbital information of the spacecraft and creates a schedule file with dates, and the general information about the pass. This file is then transferred to the automation execution tool, which contains many executable scripts called blocks that can be executed when the schedule file calls them. This tool is a skeleton of procedures. A procedure is a principle mechanism used by the operator to control the ground and space segment of a mission. It is used during functional testing in the pre-launch phase and in post-launch in-orbit operations to enable system automation when these procedures are combined with a mission automation system. An inputted schedule file fills the skeleton with information. This block system enables the PROBA team to operate extremely flexible and to add new functionality whenever it is needed without disturbing the already existing system.
Depending on the schedule file, different blocks in the automation execution tool are connected and filled with time dependency. Therefore, during a pass, the system executes the different scripts in their dependency as they were given in the schedule file.
Nevertheless, on-board of the PROBA satellites, there are highly automated systems [8], making daily operations during mission planning and passes less complicated than the operations for OPS-SAT.
A different approach to the issue is used by ESOC in Darmstadt. They developed a software called MATIS, which is short for Mission Automation System [9]. This software has been around for several years, nevertheless, it was barely utilised until recently. GAIA was the first mission using this software operationally [10]. Currently, other missions are implementing MATIS to automate their mission operations too. Missions like ExoMars, Sentinel-5p, Sentinel-2, the future Sentinels, BebiColombo and of course OPS-SAT are currently preparing for the operational usage of MATIS.
Since MATIS has only been used for large missions with a high budget, operation engineers have been watching the spacecrafts during passes up until now. Therefore, there has not been a lot of implementation of automation yet. With GAIA and ExoMars, the operation development is currently being pushed towards automation. Nevertheless, OPS-SAT will be the first mission using MATIS for completely autonomous and unsupervised passes. This makes the nominal operation automation more complicated and demanding with respect to redundancy.

1.4 Automation background at ESA

The space industry is on the upswing. Due to the growing number of satellites big established companies and space agencies like ESA are currently adapting to the need and availability of automated ground systems. Additionally, NewSpace companies and university satellite projects rise, starting and operating their own satellites. In comparison to agencies like ESA and NASA, NewSpace companies usually do not have a significant budget for spacecraft operations. Due to the high costs of ground segment equipment and operational costs during passes, these companies seek an automated system, reducing the pass costs significantly. Even though mission operation automation systems have been developed for a long period of time, these systems are not yet fully matured in the current state. This leads to unexpected behaviours during operation, which is unacceptable. For an automation system to become an established solution in the space industry, the system must run stable, reliably and trustworthy.
Unfortunately, such a system does not yet exist in full capacity yet and further research and development are needed. In addition, an adaptation of working legacy systems to full automation is a challenging and lengthy process, due to the redundant testing required to certify these systems.
OPS-SAT will represent the bridge between the current state of the art and an established operation automation solution. It will prove the feasibility and accelerate the development of such a system. If OPS-SAT succeeds, other missions will also adopt automation systems, leading to an era of ground segment automation and affordable spacecraft operations. A glimpse of further trends, future developments and additional reasons why automation is needed is caught in Section 7.5.
As mentioned in Section 1.1, operating a satellite can require a significant amount of resources and might need operation engineers available watching over the spacecraft 24 hours a day. Due to the rising automation on-board of satellites, operations become less vulnerable against anomalies on the spacecraft, since error correction can be done automatically by the spacecraft itself without the ground having to interact with it.

1.5 Goals of the Thesis

The main goal of this thesis is the automation of OPS-SAT operations during unsupervised passes and testing of the built system. To achieve this goal, several sub-goals are formulated. The investigation of different operation automation options, implementation of the chosen method, conversion of existing Excel procedures for manual operation into automatically executable blocks and testing of a set-up automation system.
Furthermore, this thesis with its appendix shall provide a deeper understanding of the operation automation, its approaches and reason of choices. It shall provide an overall

manual to enable future missions and projects to learn from this thesis and implement their own automation system based on the experience of this work. Additionally, this thesis shall represent a step towards standardised mission operation automation and shall accelerate the current shift from manual to automated operations.
A success of this thesis work will significantly reduce the operational costs and human error during operation for OPS-SAT and hopefully future missions as well.

1.6 Cubesats

The cubesat concept has been developed in 1999 by the Stanford University and the California Polytechnic State University. It is an international programme with the goal of bringing affordable small satellites into orbit. A cubesat consists of units, each ten centimetres in width, length and height and a maximum of 1.33 kg in mass. These units can be stacked to allow more payload and peripherals to be integrated into the spacecraft.

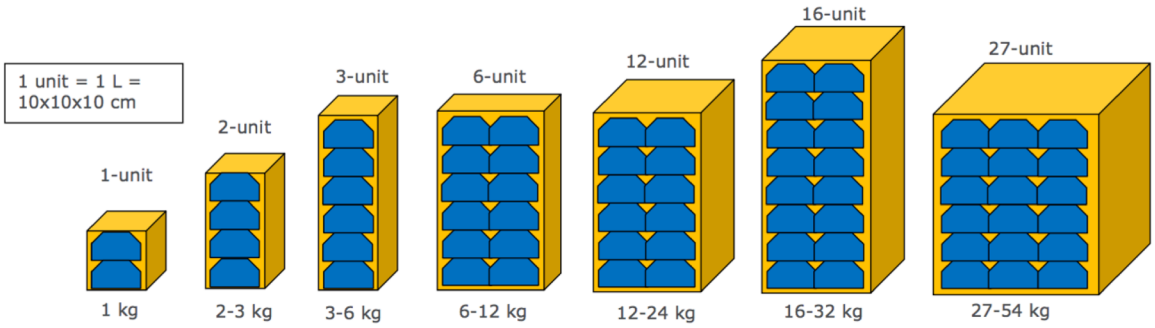


Figure 1.1: Cubesat unit dimensions [11].

The standardised size of those units results in a standardisation of its components. This is a huge advantage, since other missions can buy already tested and flight proven instruments or peripherals off the shelf. This reduces the development costs.
Nevertheless, the largest advantage of cubesats is not the hardware itself but the PicoSatellite Orbital Deployer (POD). PODs are the connection between launcher and cubesat and eject the cubesat from the launcher into orbit. These PODs are usually bought from external companies which are responsible for the very expensive and critical testing of the POD for the corresponding launch vehicle. This outsourcing of the POD reduces the time and cost for the development of a cubesat significantly and enables universities and private companies to develop and deploy their own satellite into space. Therefore, it is important to comply with the given standards to fit into a standard POD of an external party instead of building and verifying an own POD.
Because of the reasons stated above, OPS-SAT also uses a POD from an external company and is cooperating with Tyvak.

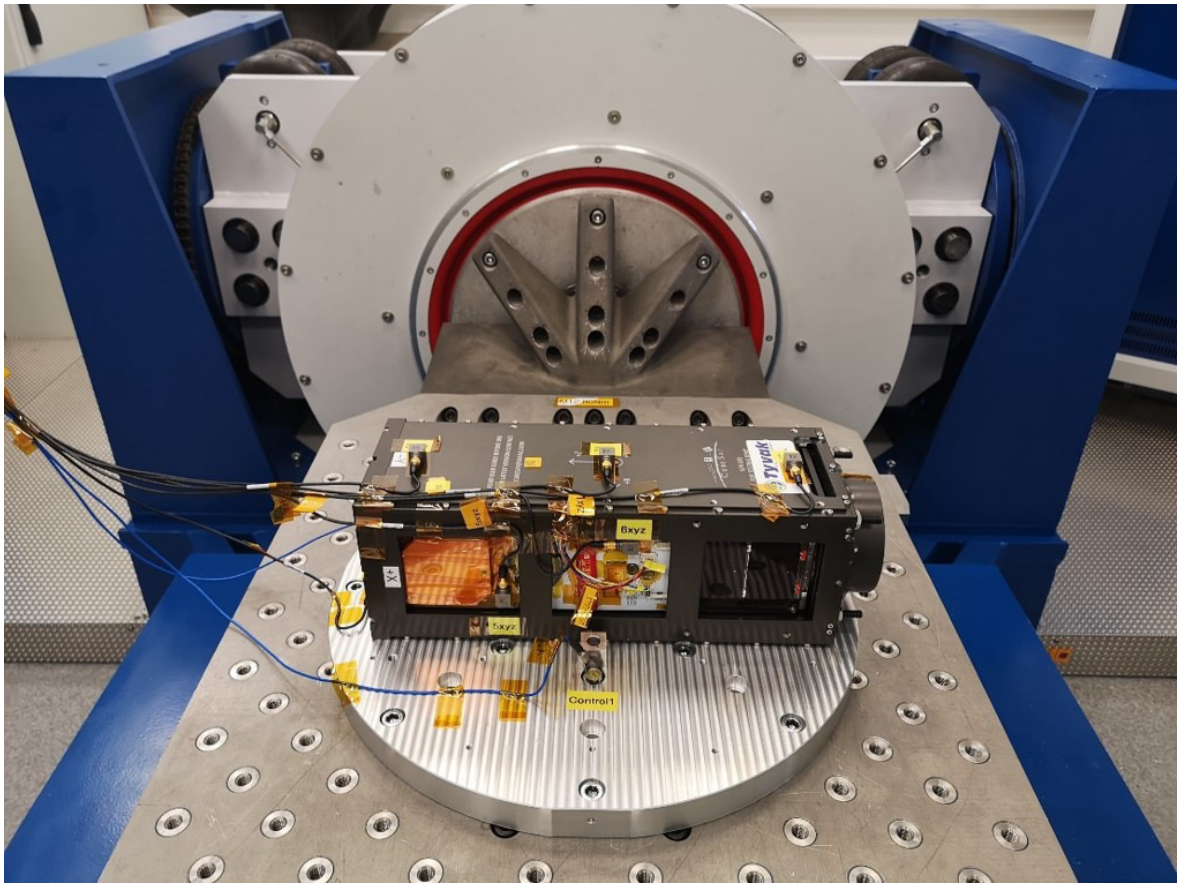


Figure 1.2: OPS-SAT POD with OPS-SAT inside, mounted on Shaker.

1.7 OPS-SAT

OPS-SAT is a flying laboratory designed to overcome long existent boundaries which are slowing down spacecraft technology significantly. It will test and validate new hardware and software, as well as new techniques in mission operation, on-board systems and complete automation.

The spacecraft consists of three units whereas one cubesat unit measures ten times ten centimetres. It can be roughly separated into two interactively connected parts. The first segment, one unit in size, is the core section of the spacecraft and designed to keep OPS-SAT safe at any given point. It includes power, OBC, GPS receiver, OBSW, FDIR, UHF communications and the functionality of the cADCS.

The second segment, two units in size, contains the payload and therefore, the experimental platform of OPS-SAT. It consists of the SDR, processing core (800MHz, 2GB of RAM, reconfigurable FPGA), X-Band transmitter (up to 50Mbps), iADCS (star tracker, reaction wheels, gyroscope, magnetorquers), S-Band transceiver (RX = 256 kbps, TX = 1Mbps), three extra reaction wheels, Camera, Photon receiver, X-Band antenna and Laser Retro Reflector.

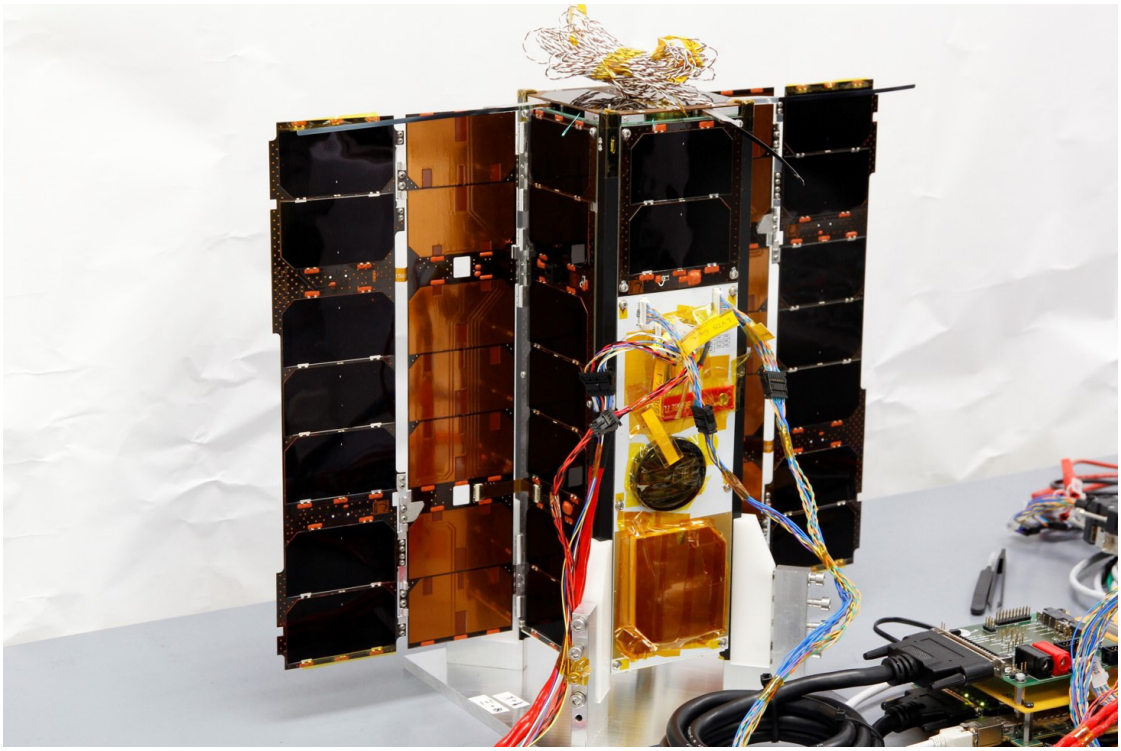


Figure 1.3: OPS-SAT Flight Model front view.

To this payload segment of OPS-SAT the experimenters can upload self-coded experiments and access all experimental peripherals, including sensors, SDR and ADCS. Further details concerning this issue can be extracted from Section 2.3.
The concept of OPS-SAT allows the experimenter to access a huge variety of components and even critical peripherals. Nevertheless, to guaranty safety of the spacecraft, a brutal FDIR approach is used. The FDIR monitors every component the experimenter can influence and switches off the experimental payload section immediately when any component value drops out of its assigned range. After the FDIR switches off the experimental section, the core section switches to safe mode, recovers the spacecraft to a safe and operable state and waits for commands of an operator during a pass.
This approach of a brutal FDIR gives the experimenters the opportunity to create risky experiments and completely new operational approaches that no one has ever dared to try on any space mission. Therefore, OPS-SAT is a unique opportunity for all nations all over the world and of course ESA itself. It allows to test and try more efficient and completely new approaches for mission operation, control, communication, modulation and sensing with no risk and no cost for experimenters.
OPS-SAT has been built to break the boundary of "Has never flown, will never fly!", accelerating mission operations, automation and space technology.

Chapter 2

Background

2.1 Programmes

2.1.1 MATIS - Mission Automation System

The EGOS Mission Automation System (MATIS) was built in support of the ground and space segment for mission operation automation.
MATIS is a ground application running on a local machine. It is located in the same network as the mission control software like SCOS-2000 as described in Section 2.1.4 which is used for daily operations.
It is intended to deploy the MATIS system [12] on the machines for:

- Preparation Environment
- Test and Validation Environment
- Operational Environment

The core functionality of MATIS consists of the following components:

- Procedure and activity automation
- Spacecraft system telemetry limit check
- Anomaly and disturbance handling control
- Housekeeping process automation
- Version control of operation automation repository

The MATIS system consists of four components:

- MATIS Server
- MATIS Central Repository

- MATIS Designer
- MATIS Operation Manager

The **MATIS Designer** is the Graphical User Interface (GUI) that is used to edit and maintain the Automation Model within the Preparation Environment. In the Preparation Environment Schedules, Procedures, System Elements, integration of the MIB and the repository structure of the Operational Environment is created, structured and modified regarding to the needs of a mission. Detailed information about the mentioned components will be given later in this section. The MATIS Designer is the only tool which also has official support for Windows. This allows operation engineers to work and design on the operation model even on their own personal computer, without having access to the mission control servers.
The **Operation Manager** on the other hand is the GUI for the operational side of MATIS. Therefore, the Operation Manager shows processes, planned operations, events and schedules running on the MATIS Server. Furthermore, it allows the user to execute procedures. In the Operation Manager, procedures and schedules can be executed manually or automatically. Additionally, the execution success status of current and past procedures and activities can be monitored as well as the log of previous actions. At last, the Operation Manager enables the user to stop the MATIS Calendar and therefore, cancels the execution of future tasks, waiting for an operator to activate the MATIS Calendar again.
The generated model and repositories are managed and distributed to the different MATIS system components by the **Central Repository**. It allows to use the MATIS Operation Manager, the Designer and the MATIS Server to be distributed to different machines and enables interconnectivity between these elements. Nevertheless, the Central Repository is only active in the Preparation, Test and Validation Environment. In the Operational Environment all the validated procedures and schedules of the Central Repository are exported and therefore, decoupled from the Central Repository to ensure a static environment during operational usage of MATIS.
The **MATIS Server** is the core of the MATIS system and responsible for compilation, execution and validation of procedures and schedules, all background processes and the communication with other interfaces.
MATIS interfaces the following systems as described in Chapter 5:

- Service Management Framework (SMF), Section 2.1.3
- Mission Planning System (MPS), Section 2.1.7

This allows MATIS to speak and interact with systems outside of MATIS and makes MATIS a tool which can be expanded to other programmes in the future. Additionally, the interface with the MPS enables not only automation of operations but also the

planning and preparation of the pass itself. The MATIS execution model is a four layer system. It handles the execution hierarchically. The following enumeration depicts the hierarchy from the bottom layer to the higher level.

- Calendar management layer in charge to schedule MAPS, MAUS and MAES for execution
- Schedule execution layer in charge of running MAPS and MAUS and their execution of procedures referenced by these schedules
- Procedure execution layer in charge to execute procedures
- Activity and external event execution layer invoking external entity services called by a procedure and forwarding events raised by the external system to procedures and schedules

Further information can be extracted from the MATIS high level architecture overview [13].

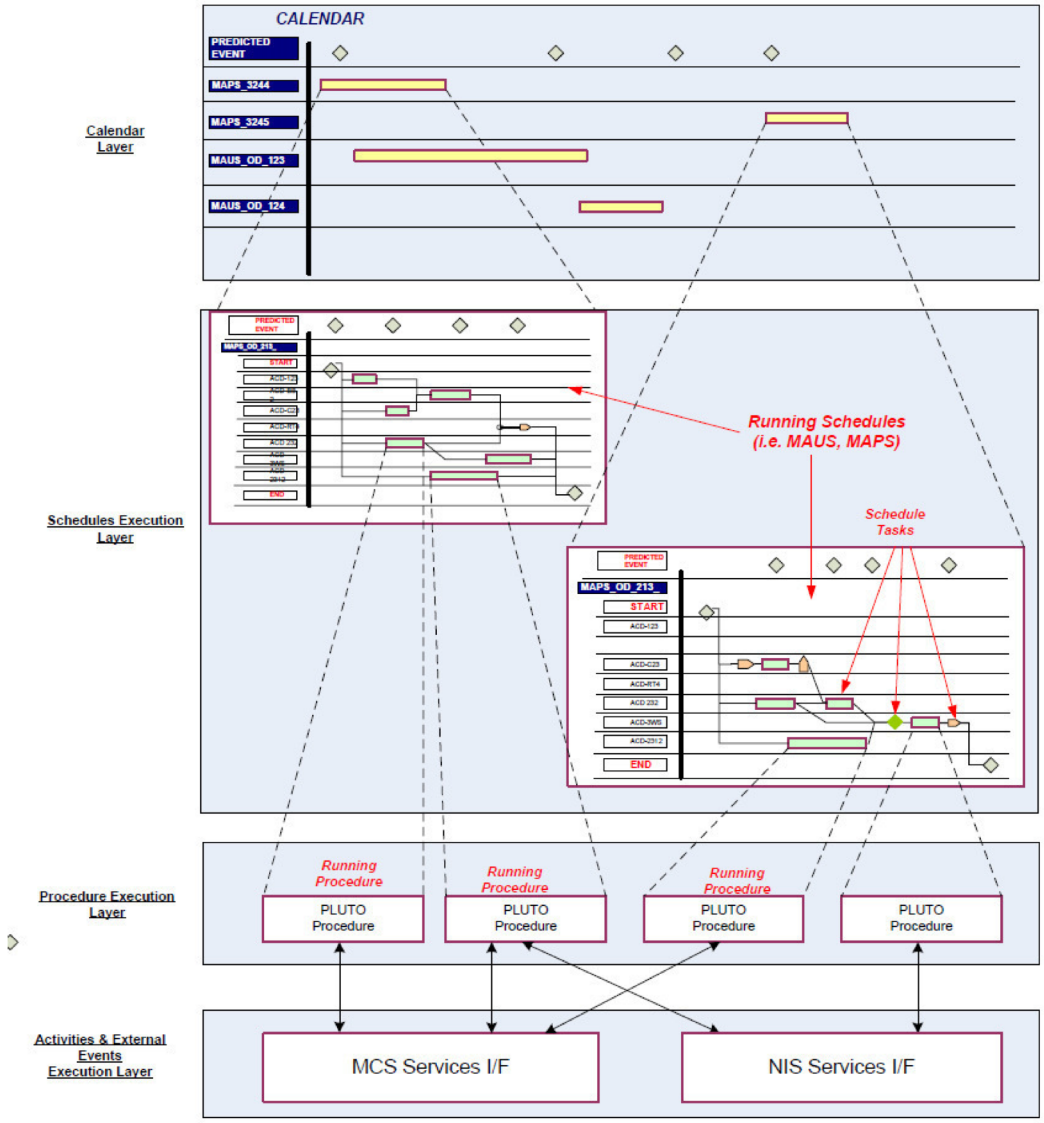


Figure 2.1: Execution environment entity of MATIS [14].

OPS-SAT Specific Functions

In OPS-SAT, many different mission specific technologies have been implemented, making OPS-SAT a test-bed for a lot of technology demonstrations. These demonstrations reach from never flown software to in-house development to software demonstration and prove of concepts. Additionally, it will be used to verify software used by cubesats outside of ESA/ESOC but which were never tested within ESA itself. An example of a programme for the last category is CSP-term (Section 2.1.8). It is used for cubesats outside of ESA by GOMSpace but has never been tested inside of the ESA framework on a flying mission.

An in-house development is the Data Proxy (Section 2.1.4) which is needed due to the various ways of communicating with the spacecraft (Section 5.2) and the needed routing of the signal to other peripherals like the Flatsat models for testing.

In addition to this, the Space Shell (Section 2.1.10), a completely new approach to communicate with the spacecraft, is implemented and tested for the first time in-flight. It allows an access to the spacecraft during live S-Band passes similar to a Linux shell. Moreover, a file management system (FMS) (Section 2.1.9) will be flown to enable fast and reliable data transfer for big files from ground to the spacecraft and other way around.
Furthermore, OPS-SAT contains a programmable FPGA which consumes 15 Watts of power while it is getting programmed. Therefore, no FPGA experiments are allowed during eclipse, conserving the batteries.
Additionally, OPS-SAT is designed to be an experimenter platform. Therefore, it can be controlled by the experimenters within their assigned time slot via experiments running on the Spacecraft Experimental Processing Platform (SEPP).

Groovy Scripts/Functions

Additionally to the procedures and activities written in PLUTO language (Section 2.1.2), MATIS provides the option to implement Groovy functions and scripts. As the name suggests these functions are written in Apache Groovy.
Apache Groovy is a very powerful language and is used by many known companies like NETFLIX, CISCO an Thales. Apache Groovy was created for Java developers to increase their productivity due to the readable and expressive syntax and its smooth Java integration. Therefore, complex functions not being able to be implemented in PLUTO language or compiled by the MATIS Server, like creating a server client application or text truncation, can be created and called by a MATIS procedure. This breaks the boundary of the restricted capabilities of the PLUTO language.

MATIS System Elements

On a high level, the automation model consists of the following seven elements forming the structure of the model.

(i) System Elements
The automation model is hierarchically built up by the System Elements. The model is composed and structured by System Elements, with no constraints on the maximum number of sub-SEs. Activities can be performed to and in a System Element.
The System Element itself can contain Activities, Reporting Data, Events and Schedules.

(ii) Activities
Activities are monitoring and control functions. Activities can be a Procedure, a Telecommand or Ground Services.

(iii) Procedures
In MATIS, procedures are written in the programming language PLUTO and can be executed by the MATIS system.

(iv) Telecommand
Activity Telecommands are linked to SCOS telecommands and can be imported from the MIB in MATIS.

(v) Ground Service
The SMF drivers provide the Ground Services directly and give access to new functionality within MATIS.

(vi) Schedules (MAES, MAUS, MAPS)
Schedules are XML files which allow MATIS to plan and execute procedures without human interaction.
Mission Automation Event Schedules (MAES) contain the events and their corresponding time of occurrence. This allows MATIS to add events to its calendar. Each MAES can also have a spacecraft ID which defines the affiliation to a specific spacecraft.
Mission Automation User Schedules (MAUS) are used to schedule tasks. Such a task can be used to check for OoL every morning at a specific time or preparing the ground segment for an upcoming pass and command the spacecraft. They contain a list of tasks which themselves can have an execution condition or dependencies to the previous tasks.
The Mission Automation Planned Schedule (MAPS) is similar to a MAUS. The difference is their generation. These Schedules are generated by the Mission Planning System (MPS) and are therefore, automatically generated.
General information is displayed in [15] whereas structural information about the MAUS and MAES can be extracted from [16] and [17], respectively.

(vii) Operational Mode

The Operational Mode is designed to guarantee a safe and well tested environment during automatic commanding with MATIS. To achieve this, the MATIS Server and the Operation Manager are cut off from the SVN server and are not able to communicate with the SVN repository and possible changes.
Instead of the SVN repository, the MATIS Server gets its procedures from a dedicated compressed Tag. A Tag is a snapshot of the current repository which can then be exported as a compressed folder to the MATIS Server in Operational Mode. The

MATIS Server takes this Tag and uses only the procedures, schedules, drivers and Groovy functions which were exported into the Tag. Therefore, the system is static. To make this procedure as safe as possible, the default setting when exporting a Tag, is advised to only export the validated elements. Since elements have to be set as validated manually by the operation engineer, these procedures are well tested and thus, can be considered as safe.
Nevertheless, this function can be deactivated, enabling exported Tags to contain draft and validated elements. The use of this function shall be avoided, but can be used when a bug in MATIS Editor occurs where a functioning procedure is not being able to be set to "validated".

In Operational Mode, the IN-TRAY function is activated, which allows MAES and MAUS to be executed automatically when the files are placed into the dedicated IN-TRAY folder in MATIS Server as seen in Figure 5.2. As it will be explained in further detail in Section 5.1, the MAES shall be dragged into the IN-TRAY folder first. When the MAES is loaded, its events are entered inside of the MATIS calendar. After this, the MAUS can be placed into the IN-TRAY folder allowing it to synchronise its tasks with the corresponding events existing in the calendar.
Note: It is important to fulfill this order, since tasks from the MAUS only synchronise with events already existing inside of the calendar.

2.1.2 PLUTO

The "Procedure Language for Users in Test and Operations" (PLUTO) is a computer language specifically designed for mission operations. The language itself is designed to be read, not to be written in particular. A main difference to other programming languages are the dependencies. Whereas other programming languages use the path annotation to imply dependencies, such as "first/second/third/element", the PLUTO language uses the English grammar to give dependencies and therefore, results in "element of third of second of first" for this example. Therefore, PLUTO is verbose and provides important information in an understandable way, without knowing the exact rules of the language itself. This allows mission operation engineers to get a very quick overview of the executed code.
Since it was developed for mission operations, PLUTO is designed to program procedures that are divided into different bodies which can be seen as building blocks, structuring the procedure. An illustration of this principle is depicted below.

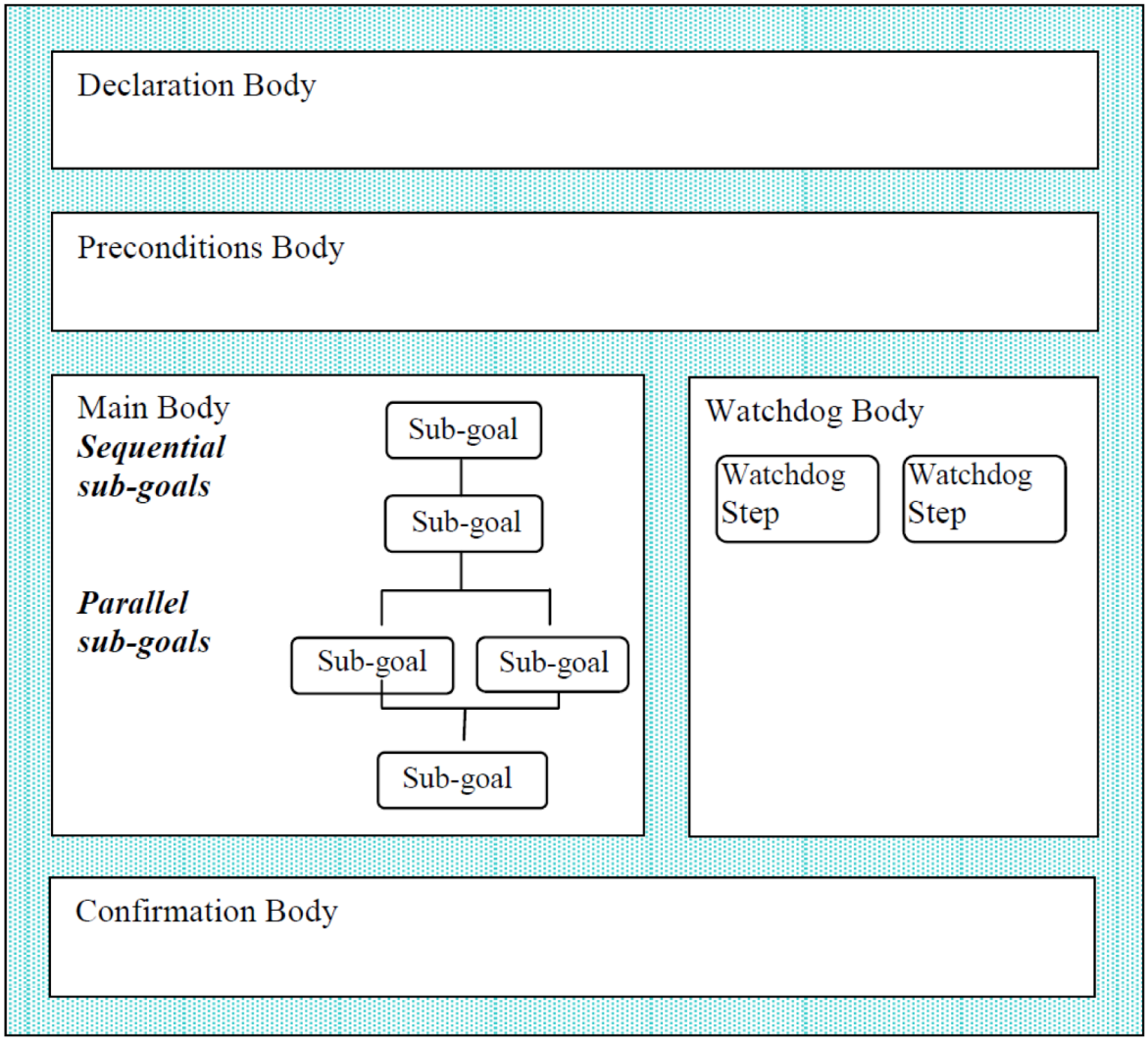


Figure 2.2: Example of a procedure and its elements [12].

PLUTO is a goal oriented programming language, which means that executable parts of the code are divided into steps. Each step contains commands, TM checks or many more possible actions. Each action is labelled after execution. The label contains a parameter indicating whether the action was successful or not. At the end of a step, all indicators of actions inside are checked. If this confirmation status equals to "confirmed", a step is considered successful and the procedure continues its execution. If the confirmation status does not turn out as expected, the procedure aborts, cancelling its further execution at the end of this step. Nevertheless, sometimes a failure of an activity is expected. Therefore, PLUTO has the functionality to configure its confirmation status depending on a user defined if condition.

```
// STEP: 3, OPERATION: DEFINE AND ENABLE AGGREGATIONS
        initiate and confirm step DEFINE_AND_ENABLE_AGGREGATIONS
            preconditions
                if raw_value of CMD_TM_LINK of MISC_Variables of SSM = "CONNECTED"
            end preconditions

            initiate MG_DH00 of AggregationDefinitions of MIB_TCs of Telecommands of SSM;          //Aggregation DH00
            wait for 0.5s;
            initiate MG_PP01 of AggregationDefinitions of MIB_TCs of Telecommands of SSM;          //Aggregation PP01
            wait for 0.5s;
            initiate MG_PP02 of AggregationDefinitions of MIB_TCs of Telecommands of SSM;          //Aggregation PP02
            wait for 0.5s;
            initiate and confirm M040603b of NanomindTCs of MIB_TCs of Telecommands of SSM         //MC  enableGeneration
                with arguments
                    raw value of LEN1 := 3,     //TYPE: Unsigned integer    //DESCRIPTION: Length field
                    ~04060300 := "DH00",        //TYPE: Unsigned integer
                    ~04060301 := "1",           //TYPE: Boolean             //DESCRIPTION: enableGeneration
                    ~04060300 := "PP01",        //TYPE: Unsigned integer
                    ~04060301 := "1",           //TYPE: Boolean             //DESCRIPTION: enableGeneration
                    ~04060300 := "PP02",        //TYPE: Unsigned integer
                    ~04060301 := "1"            //TYPE: Boolean             //DESCRIPTION: enableGeneration
                end with refer by enableGeneration_0;
            confirmation
                if (confirmation_status of enableGeneration_0 = "confirmed")
            end confirmation
        end step;
```

Figure 2.3: Example of a step with precondition and explicit confirmation condition.

The PLUTO language standard and all its functionality in its full complexity is defined and explained inside of the ECSS standard ECSS-E-ST-70-32C [12].

2.1.3 SMF - Service Management Framework

The Service Management Framework (SMF) enables the MATIS user to interact with other programmes and with MATIS itself through SMF drivers translating between the programmes.
SMF is designed as a service provision middleware infrastructure. Together with SMF drivers, it connects to the Application Unit (AU) of an external programme. An AU can be viewed as an input or output port into a programme which feeds or gets information to or from the programme to the Service Management Service.
The service can be extended by additional drivers. Due to this functionality, the SMF itself is generic and can be connected to any programme and therefore, connecting individual programmes into an interactive network. This enables separate systems to communicate with each other without specifically being designed for this purpose. In the case of OPS-SAT, this system connects SCOS, MATIS, REALS, SwissKnife and the Network Interface System (NIS), creating a powerful tool for automation of operations.
The SMF service is based on the standard ECSS-E-ST-70-31 (Monitoring and control data definition) [18]. Using a standard for this interface guarantees the possibility to model services in a higher level way, independently from the low level protocols. Additionally, it enables a standardised method to write services and to access them. More information regarding the functionality of SMF with respect to MATIS is presented in the MATIS Interface Control Document [19].

2.1.4 SCOS - Satellite Control and Operations System

EGOS SCOS-2000 (SCOS) is the used mission control software. Further information about full functionality is given in the documents starting with ID S2K. A first guide is given in [20]. In the context of this thesis, SCOS can be seen as a slave programme, getting input from MATIS over the SMF and respectively handling the TC and TM management to and from the spacecraft.

Application Launcher

In the Application Launcher, the different applications and services of SCOS can be controlled. The applications can be started, stopped and restarted by the colourful buttons in the bottom left corner. Additionally, the buttons Log and Conn Status can respectively be used for logging and for the connection status of the application. By going through the tabs on the top of this Application Launcher, one can find the applications and services sorted by their category.

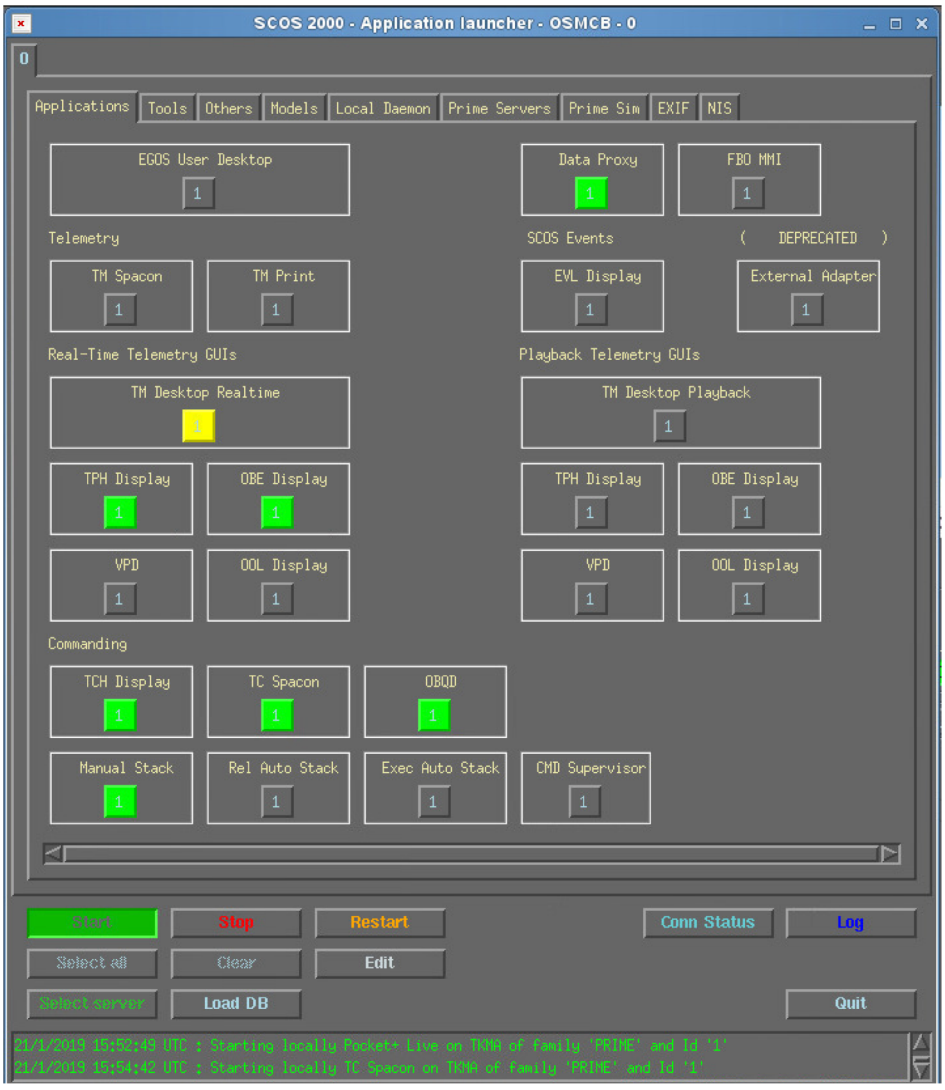


Figure 2.4: A screen capture of the SCOS - Application Launcher window.

Manual Stack

In the Manual Stack, commands and stacks can be loaded, armed and sent. Commands are used to control the spacecraft and to talk to it. Stacks are lists of saved commands with their order, execution time and values. In the manual stack, different options can be set, controlling the permissions when a command can be send. These include the option to send commands with or without telecommand link and with or without telemetry received from the spacecraft.
In normal operation, commands should always be sent with telemetry and telecommand link to the satellite to ensure telecommand execution verification on ground, after a command has been successfully executed on the spacecraft.
Sending commands without a link requires knowledge about the satellite position and its hard-/software system and its response, since the command is blindly sent when this option is enabled.

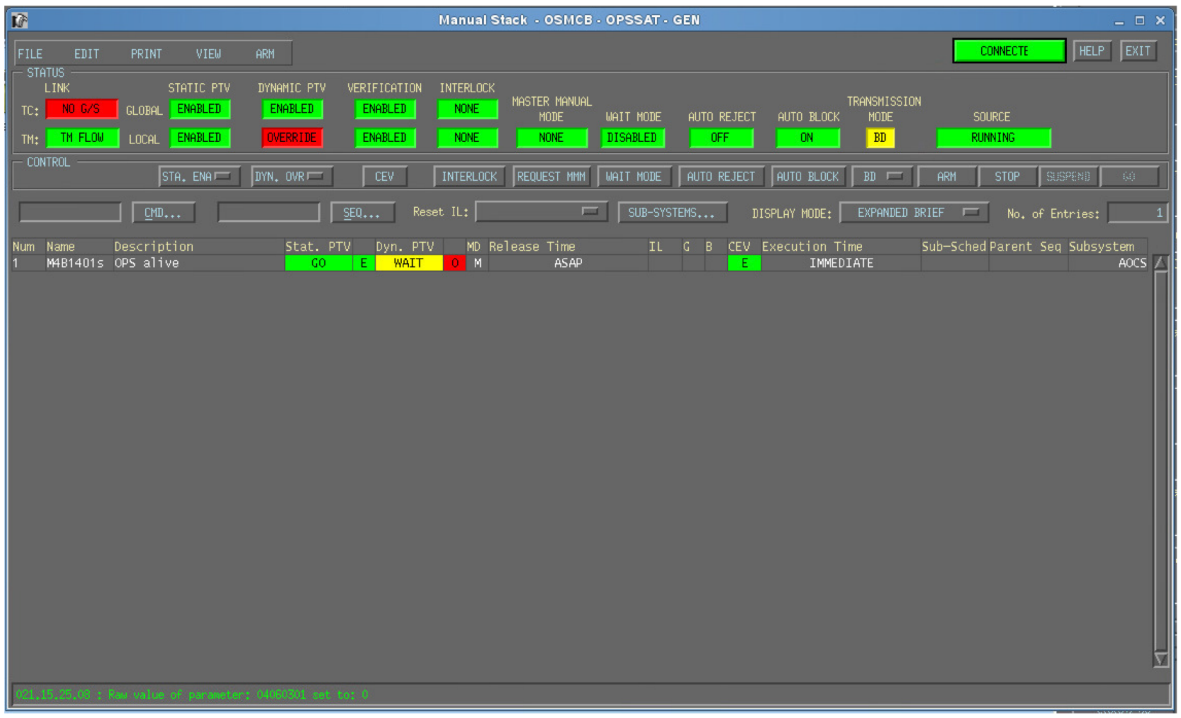


Figure 2.5: A screen capture of the SCOS - Manual Stack window.

Data Proxy

The Data Proxy works as an alternative access point from MCS to telemetry sources. During nominal S/X-Band operations, access will be used with the NIS via Space Link Extension (SLE) [21]. Data Proxy is used to access the UHF chain, the FlatSat, the CAN bus directly (via TUG Adapter) and CORTEX direct access for high speed CFDP [22] uplink/experimenter access. During operation this configuration changes slightly.

Like the On-board Software, Data Proxy itself is a Mission Operation (MO) Services provider [23, 24] and can be commanded from SCOS.

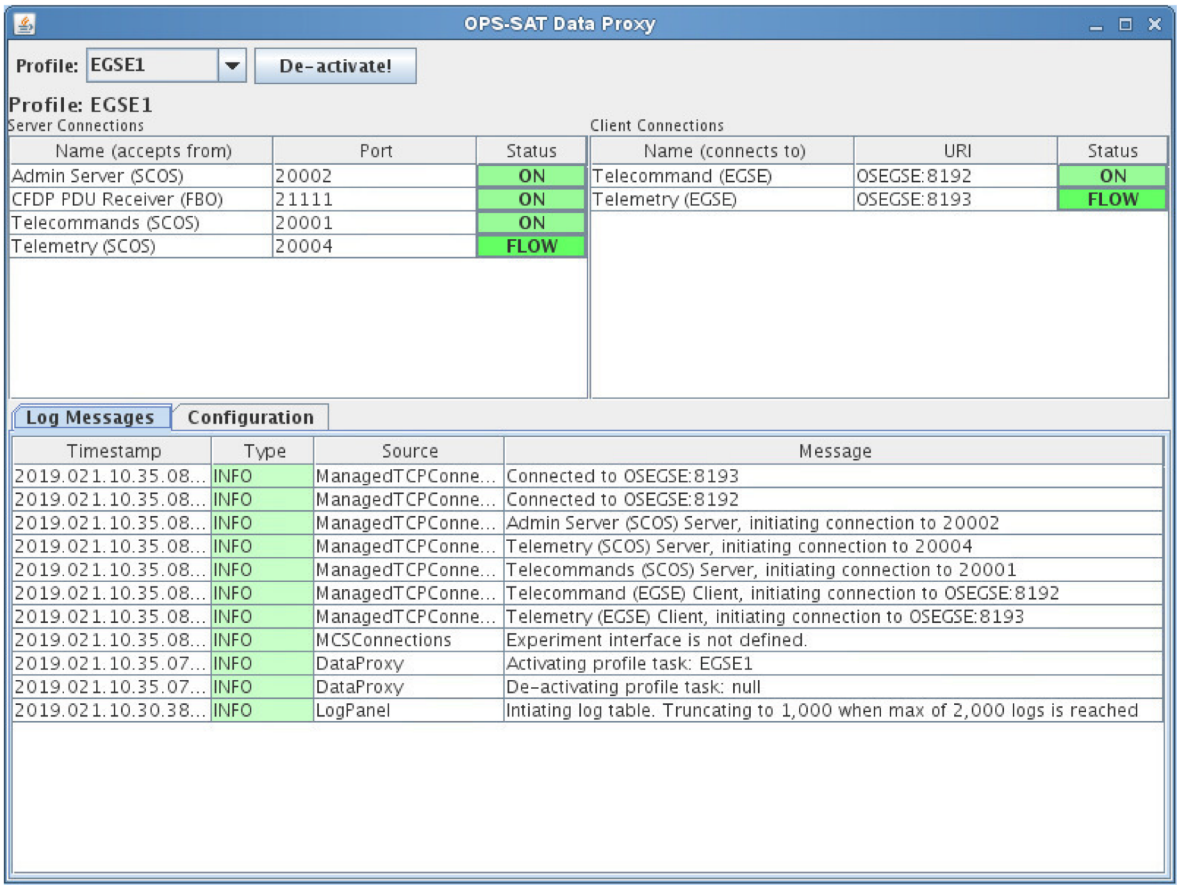


Figure 2.6: A screen capture of the SCOS - Data Proxy window.

TC Spacon

The TC Spacon is responsible for the correct encoding of the telecommand packets and the overall configuration regarding the sending of telecommands.

Event Display

The event display shows all events arriving at SCOS. These events can be raised by internal functions within SCOS or as incoming packets from the spacecraft. As an example the activation of a specific hardware component on the payload side can be seen in the event display and signalises the correct execution of a command.

In-Depth Information

Due to the complexity of SCOS, only the most basic and most used features were shallowly explained above. For detailed information about all functionality of SCOS, the following figures are provided. Those pictures shall give an overview of the internals

used in SCOS and where to find each. All documents can be found via this link file://esaad/esoc/OPS-GI-CurrentRef/SCOS-2000/SCOS-2000%20R5.4.21/Documentation/S2K/index.html when connected to the Office Lan in the ESA internal network.

Figure 2.7: A high level SCOS documentation overview [25].

Three main documentation categories play a decisive role for system level understanding: Software Design Documents, Software User Manual and Interface Control Documents. Therefore, their main document names and explanation are listed in Appendix C.

2.1.5 NMF - NanoSat MO Framework

The NMF is a framework which allows users to deploy their applications on a spacecraft, accessing the peripherals like the camera with high level commands such as "take a picture". This makes it easy for experimenters to develop their own applications for OPS-SAT without having to know the commands on an embedded level.
To enable compatibility with ESA mission standards, NMF is based on CCSDS Mission Operations services [26].

2.1.6 Pass Prediction

The Pass Prediction is a Python programme programmed by Andre Lofaldli (ESOC/ESA). The input is a two line element (TLE) taken from NORAD which describes the orbit. It uses an open source library such as PyEphem to compute the position and velocity of the spacecraft.

From a location on ground (ESOC or TUG), the programme generates a list of events indicating the beginning and the end of a pass. These events are used to generate an XML file with the format MATIS expects.

2.1.7 MPS - Mission Planning System

The Mission Planning is responsible for four tasks: The creation of Stacks for SCOS, the MAES, the MAUS and shifting the correct files into a folder whose content will be uploaded to the spacecraft. These contents can represent images for the SEPP or experiments.
The Mission Planning gets the AOS and LOS times in an XML file from the pass prediction/planning. With this pass data, the Mission Planning will create additional events with their corresponding times, depending on the executable tasks and resources available. These are stored in a new MAES. Additionally, it will fill a predefined User Schedule (MAUS) which will then be transferred to a specific folder in MATIS, called IN-TRAY, to enable automatic loading and execution of the MAUS.
The system is designed in a way that allows the Mission Planning to also be substituted by an operator.
In conclusion, the Mission Planning System generates and prepares all necessary files for MATIS. This enables a working and easily accessible automation system.

2.1.8 CSP-term

CSP-term is a programme for low level system access on the spacecraft via UHF.
The signal is sent directly to the NanoCom where the corresponding command is routed to the peripherals.

2.1.9 FMS - File Management System

The File Management System (FMS) can transfer and synchronise data between the ground and the spacecraft. It can also create and delete folders on the spacecraft and synchronise specific files from the spacecraft to ground and the other way around.

2.1.10 SpaceShell

Space Shell is a C application developed by Benjamin Fischer (ESOC/ESA). It allows experimenters or the flight control team to communicate with the OPS-SAT Satellite Experimenter Platform (SEPP) like a Linux Shell and consists of a ground application (ground port) and a space application (space port).
Usage of the SpaceShell requires live communication with the satellite. In case of OPS-SAT, it is only used in S-Band due to band width limitation in UHF and the

missing X-Band uplink possibility.
OPS-SAT will be the first ESA mission with live shell access control of a spacecraft in orbit.

2.1.11 REALS System

The REALS system has been developed to enable operation engineers to being notified via SMS or email when telemetry parameters of the spacecraft are out of their range or certain events are triggered inside of the Mission Control Software (here SCOS).
REALS consists of two parts, the client and the server side. The REALS server is located on a dedicated machine with access to the internet as well as hardware to send SMS. Therefore, this network is considered to be "unsave" in the Operational Network in ESOC. This means this machine can only be accessed from outside but cannot access other machines. This is necessary to keep threats and hacking approaches away from the critical machines in the Operational Network, where missions operation software is running and controlling spacecrafts. Nevertheless, this security policy comes with a downside. This machine is not able to ask for data from more secure networks like the machines where MATIS or SCOS are running on. Therefore, files have to be sent to the REALS server which processes the incoming data later on.
The second part is the REALS client that consists of scripts to watch over the programmes, the telemetry and events. In the MIB, which is the data base of a space mission, value ranges for TM data are set. Two different TM limits have to be differentiated, soft and hard limits. The soft limits' purpose is to warn the operator about values which are still in acceptable range, but indicate entering a soon to be dangerous state. When the value rises or falls even further, the hard limit is triggered, indicating this value to be out of hard limits with a flag. Whenever one of these limits are exceeded, a notification is sent to the REALS server. Depending on its configuration, the REALS server triggers an action to notify the operation engineers or not.

In the REALS system, it is possible to setup different configurations, users and notification groups. Thus, different users or operation engineers can be notified for specific events or exceeding the TM limits. When an SMS is sent to the operation engineers, they can reply to this message, confirming that the SMS has been read. Additionally, a chain of notifications can be configured. This means when an operator has not replied to the SMS after a certain amount of time, a second operator will be notified. Furthermore, different shifts can be set on the REALS system for each user. Like this, the system only sends SMS to the people currently available in this shift. This enables an efficient workflow and avoids unnecessary SMSs during the night and weekend.

2.2 Hardware and Infrastructure

2.2.1 Flight Model - FM

As previously mentioned, the Flight Model consists of a three unit cubesat, whereas two units are reserved for the payload section and one unit for the core section.
The payload section contains the following components. On the bottom of the spacecraft the camera, Photon receiver, X-Band antenna and Laser Retro Reflector are positioned. At the next layer the three extra reaction wheels are placed. This is followed by the S-Band receiver with a maximum speed of 256 kbps and a transmitting speed of up to 1 Mbps. Further up the Fine ADCS (iADCS) with its star tracker, reaction wheels, gyros and magnetorquers can be found. On the next level the X-Band transmitter, which has a transmitting speed of up to 50 Mbps, is placed. Furthermore, the CCSDS TC Decoder/TM Encoder is available above the X-Band transmitter. The processing core with its powerful 800MHz processor, running Linux/Java with two GB of RAM and a configurable FPGA, as well as the Software Defined Radio (SDR) are located just below the core section of the satellite.
In the one unit core section the power units, batteries, the on-board computer (OBC) as well as its on-board software, the GPS receiver, the very important FDIR and the UHF peripherals can be seen and provide the traditional ESOC satellite system for secure and very safe spacecraft operations.
A visualisation and the exact position of the hardware can be seen in Figure 2.8.

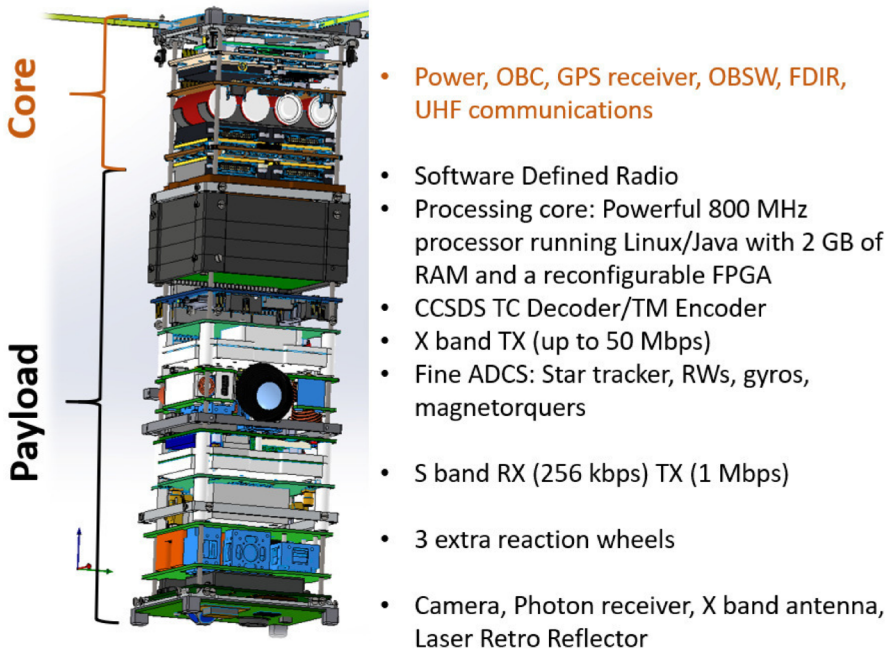


Figure 2.8: OPS-SAT Flight Model hardware (Figure provided by TUG).

2.2.2 Flatsat - Engineering Model

The Flatsat unit is almost an exact copy of the satellite peripherals. In comparison to the compact and vertically stacked flight model, this unit is mounted on a flat board for easy accessibility, debugging and switching of components.
The differences to the Flight Model are missing solar panels, second NanoMind from GomSpace, S-Band radio and X-Band transmitter. Nevertheless, even though the solar panels are missing, the Array Conditioning Units (ACUs) are present. Thus, solar array voltages can be simulated and the regulation and behaviour of the ACUs and Flatsat itself can be tested. Additionally, despite the missing S-Band radio, S-Band can still be used due to a direct connection through the EGSE.

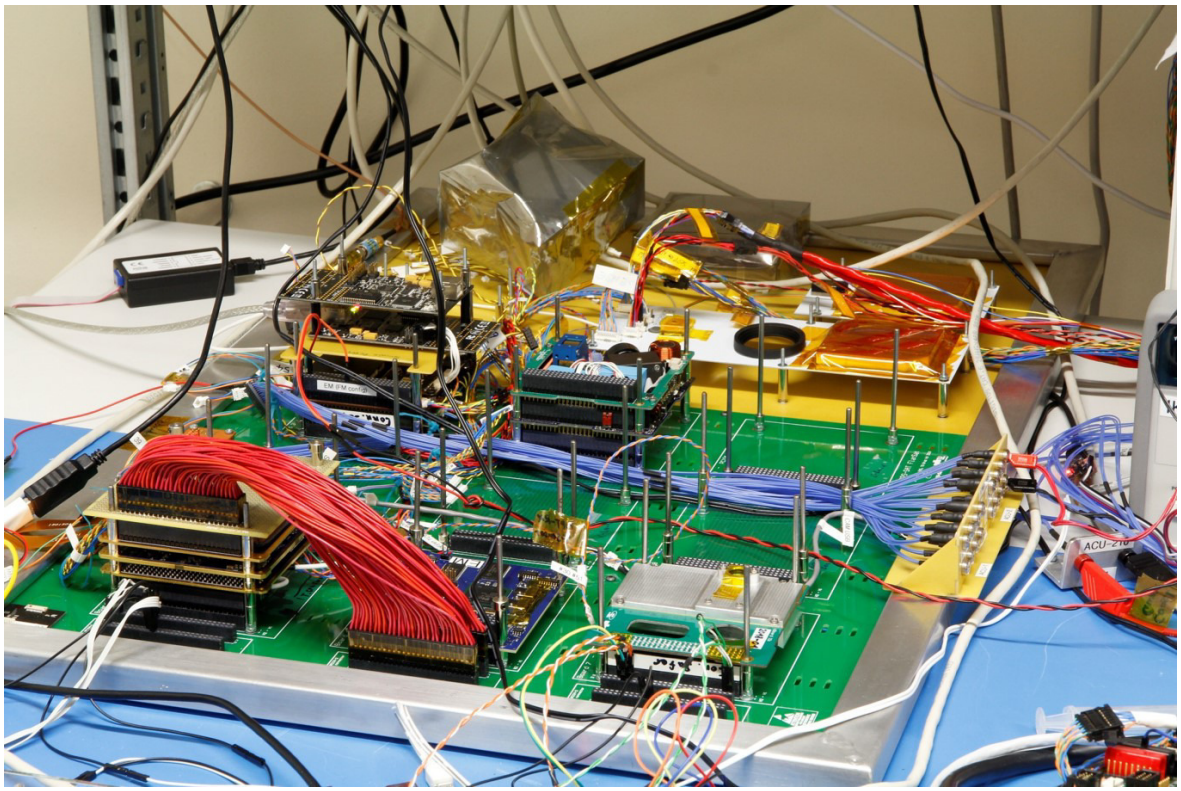


Figure 2.9: OPS-SAT Flatsat - Engineering Model (Figure taken by TUG).

2.2.3 SMILE - Special Mission Infrastructure and Lab Environment

For OPS-SAT, SMILE functions as mission control room, server room, infrastructure node and Flatsat laboratory. In Figure 2.10, the dedicated hardware is depicted. On the left, the rack for the Flatsat and power and signal peripherals are shown. On the right hand side, the server and network rack can be seen. These illustrations show the exact hardware used in great detail.

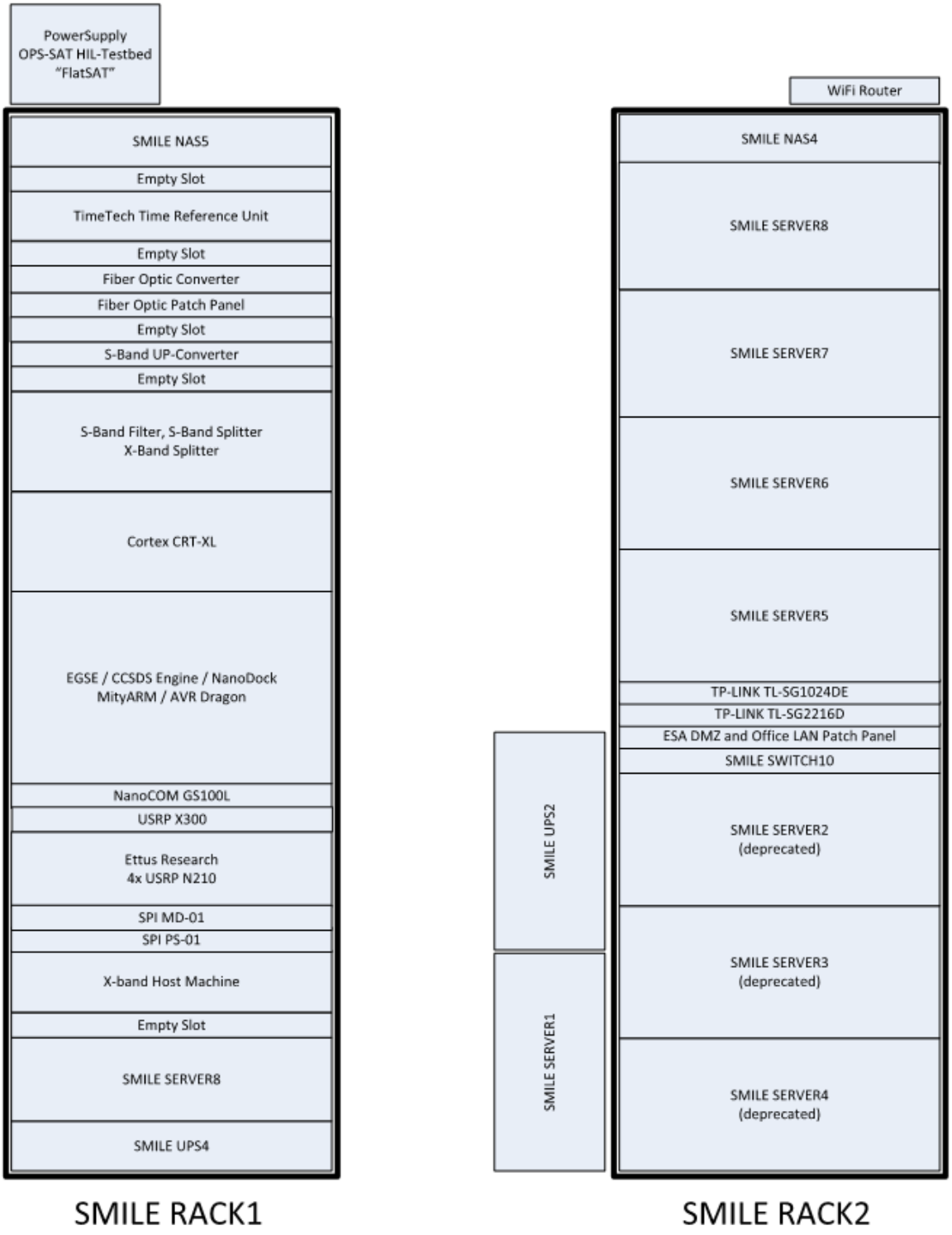


Figure 2.10: SMILE racks with peripherals [27].

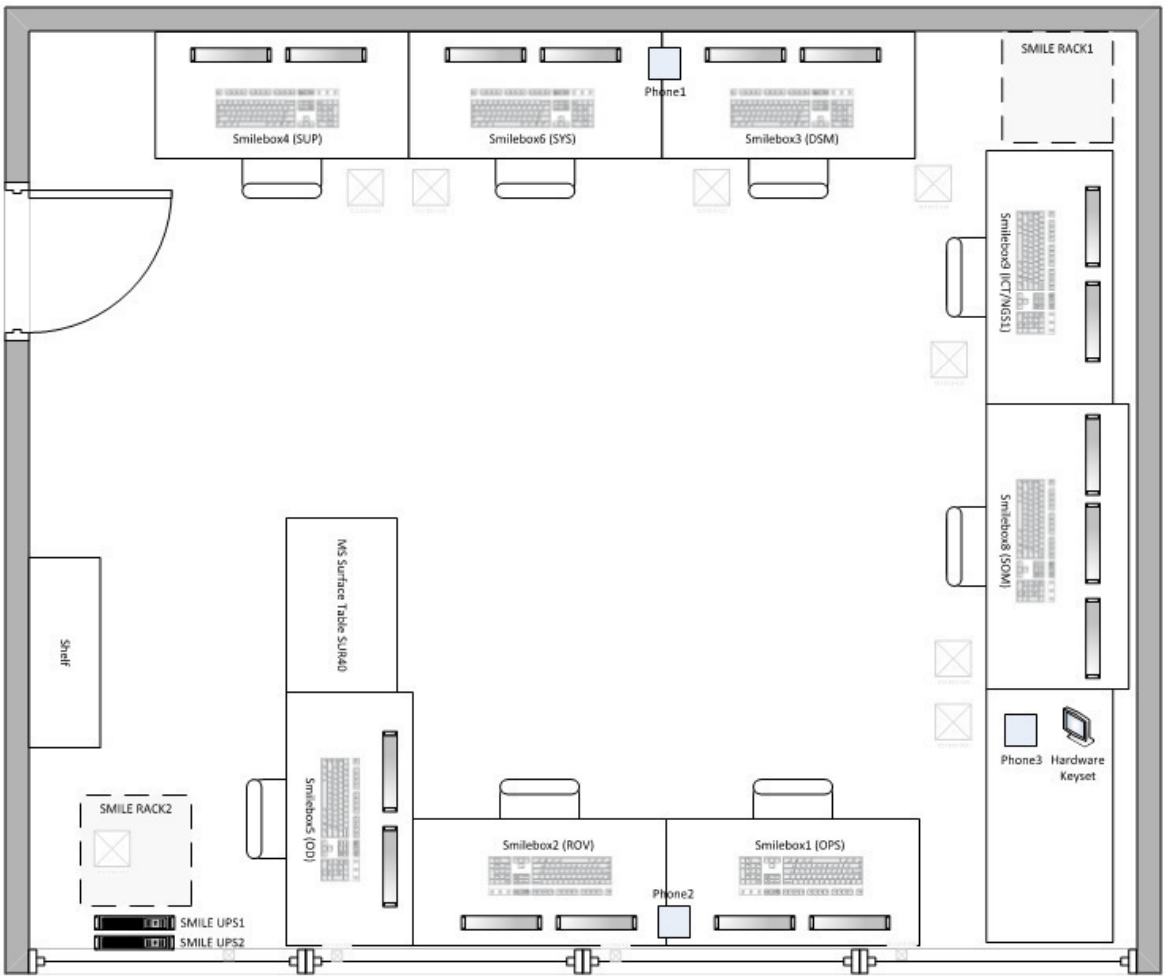


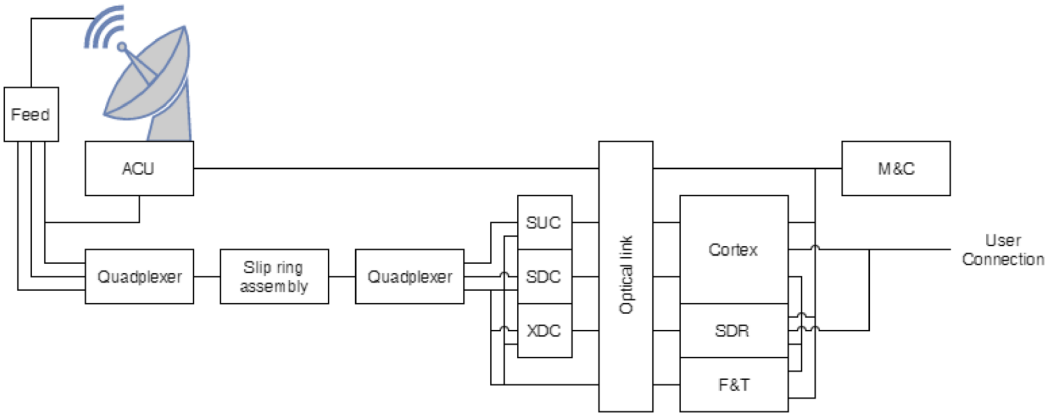
Figure 2.11: SMILE floor layout [27].

Antenna Systems

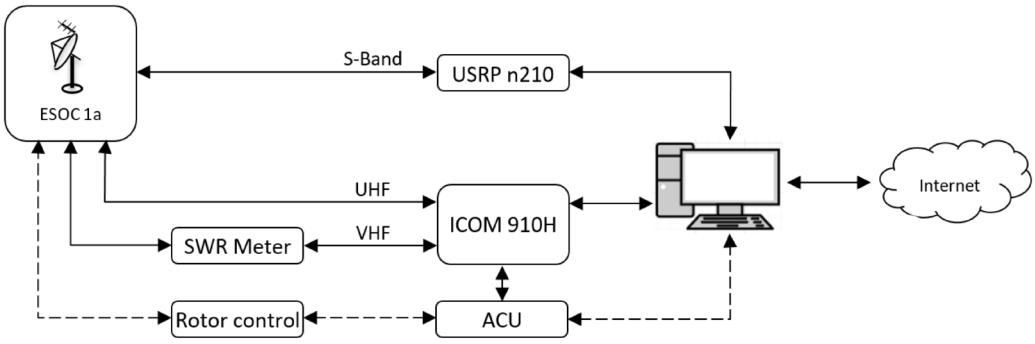
Additionally, SMILE consists of two antenna systems ESOC-1 and ESOC-1a.
ESOC-1 is capable of transceiving S- and X-Band and has a diameter of 3.7m and a maximum EIRP of 86.5 dBm in S-Band. Furthermore, ESOC-1a is capable of transceiving VHF, UHF and S-Band and is using a flexible SDR for TM/TC standards and data coding scheme. ESOC-1a shown in Figure 2.12b is mostly used for UHF in the case of OPS-SAT and as a backup for S-Band. Nevertheless, ESOC-1 depicted in Figure 2.12a is the standard antenna for both S-Band and X-Band.
To operate these antennas, a network infrastructure has to be created in which the data can be transmitted to the mission control software or data storage servers. Information about the exact network and the differences of the antennas can be extracted from Figure 2.13.

(a) ESOC-1 [28] (b) ESOC-1a [27]

Figure 2.12: Antennas for OPS-SAT.



(a) ESOC-1



(b) ESOC-1a

Figure 2.13: Functional data path diagram for different antennas [27].

Virtual Machines Setup

In operational mode, two machine chain pairs are used for redundancy. The first pair is called OSMATA and OSMCA, OSMATB and OSMCB are the second virtual machine set. Both machine sets have the same setup and it is possible to switch between the chains in case of failure, which results in warm redundancy.
The MATIS Server and Operation Manager are running on OSMATx. SCOS, the NIS, the Prime Servers and SMF are running on OSMCx. Additionally to those machines dedicated for operation and automation, many other machines with specialised purposes are available. The specialised machines reach from direct access to the satellite for debugging to a machine dedicated for experimenters to machines for configuration and automation of several ground station peripherals.

2.2.4 NIS - Network Interface System

The Network Interface System (NIS) was developed to provide a link between the Mission Control Systems (MCS) and Ground Stations all over the world. It is using the Space Link Extension (SLE) services. Additionally, it provides an External Interface for NIS clients like mission automation systems.
The NIS is composed of four components: The NIS Manager, the NIS Server(s), NIS HCI acting as a Graphical User Interface (GUI) and the NIS Configuration Tool. The exact functions and dependencies can be extracted from [29].

2.3 Experiments and Experimenters

As mentioned earlier, OPS-SAT is all about testing, accelerating the development of future space technologies, and provides a test-bed which is unique and powerful. It aims to inspire universities, researchers and companies to implement and test new technology on a real satellite with no costs or risk.
OPS-SAT provides an online platform where interested entities can sign up for. On this platform, further information on the satellites peripherals, current status and on how to programme, build and test an experiment is provided. Additionally, the web interface contains a forum, allowing experimenters to communicate and support each other. During the experiments, the experimenters have access to a light-weight mission control system. Thus, they can monitor the satellite in real time, as shown in Figure 2.14.

Figure 2.14: An example of the experimenter light-weight MCS [11].

Chapter 3

Preparation

During this thesis, one of the major elements of the work was the preparation of all the automation systems and their interconnectivity.

3.1 MATIS

3.1.1 Bug Localisation and Correction

The following section discusses the discovered bugs in MATIS during preparation and testing. Therefore, this paragraph is closely connected to the chapter Testing 6.
MATIS was developed internally by ESA and barely used in the past, therefore, the user community is accordingly small. Even though the MATIS team is working hard to develop a flawless and well tested system, bugs can enter the software without being noticed. With the help of missions actively using MATIS like GAIA and OPS-SAT, these bugs can be identified and reported back to the developers. This loop will increase in the future with missions like ExoMars, BebiColombo and the Sentinels joining the family of MATIS users.
While working with MATIS and SMF in combination to control and command SCOS, several minor but also major mission critical bugs were found. These bugs were then reported to JIRA, a software from Atlassian [30] and used by ESA to track software system progress and reported issues. Especially major issues have been reported to JIRA, helping MATIS support to identify the problem and other missions to get a stable and flawlessly running software. Smaller issues have often be solved with workarounds, since a report in JIRA can slow down other issues that have to be worked on. Many of the occurring problems, their solution and status can be found in Table 6.1 in Chapter 6.

3.1.2 SMF Drivers

As mentioned in Section 2.1.3, the drivers represent the link between MATIS, other programmes and even MATIS itself. Each SMF driver connecting to MATIS needs to be connected to the Application Unit (AU) of the corresponding programme to transfer information. The driver translates the output from MATIS into a readable format for the AU parsing its arguments into the corresponding programme. From the user's point of view, a driver combined with SMF is the interaction gateway between MATIS and other programmes.
Nevertheless, in the case of OPS-SAT, these drivers and their corresponding SMF configuration and files were not configured correctly and a lot was missing or had to be changed in order to work with OPS-SAT's environment. A detailed step by step tutorial to integrate an already existing driver into the SMF and MATIS is given in Appendix A.
This section will focus on drivers which were either newly implemented due to lag of functionality, adapted due to speciality in the OPS-SAT mission environment or were wrongly configured inside of MATIS.

Newly implemented drivers are:

- NIS driver (refer to Section 3.2.1.)
- Application Start drivers (see Section 3.2.1.)
- Calendar Driver for MATIS Calendar
- Packet Provision Driver
- TM Provision Driver

3.2 SCOS

The preparation of SCOS included the setup of new application descriptions for the task manager to start apps from MATIS, the implementation of the NIS drivers with their configuration in the SCTL, the configuration of TKMA file in SCOS to get a new SMON server and the setup of the MMIconfig to add clickable buttons in the application manager.
The first two preparations will be explained in Section 3.2.1 below.
Editing the TKMA file, which is a file where programme names, configuration and ID are stored, allows to implement new applications. In this file, the SMON server was added. The SMON server is a server which takes input from a file archive and provides the values to the application that requests the parameter and is connected to

the SMON server. This is required for TM acquisition in MATIS.
Furthermore, the MMIconfig, which stores the visual appearance of the application launcher, was changed due to comfort changes and easier debugging. A button has been added for the SMON server which indicates its execution status with a colour. The colour green displays a running server, the colour yellow an unidentified state and red a stopped application. Additionally to this, the functionality of pressing the log button for this application is now possible.

3.2.1 NIS Drivers

The SMF NIS drivers are necessary to communicate and start the NIS in SCOS from MATIS. Nevertheless, two functions were missing: First, the capability to start and stop the NIS application within SCOS itself without MATIS involved (see link "3" in Figure 3.1) and second, the SMF driver to control the NIS from MATIS through SCOS (see links "1" and "2" in Figure 3.1).
To enable SCTL to start the NIS applications, new configurations were added by editing the configuration file of SCTL (System Control Display) with the right programme names, representative for the application itself. This action connects link "3". For link "1" and "2", the drivers to start applications from MATIS were already deployed and did not have to be changed. Nevertheless, the SMF driver to control the NIS and its peripherals itself was missing. Therefore, this driver has been implemented both on the MATIS side, as well as the SCOS side of SMF.

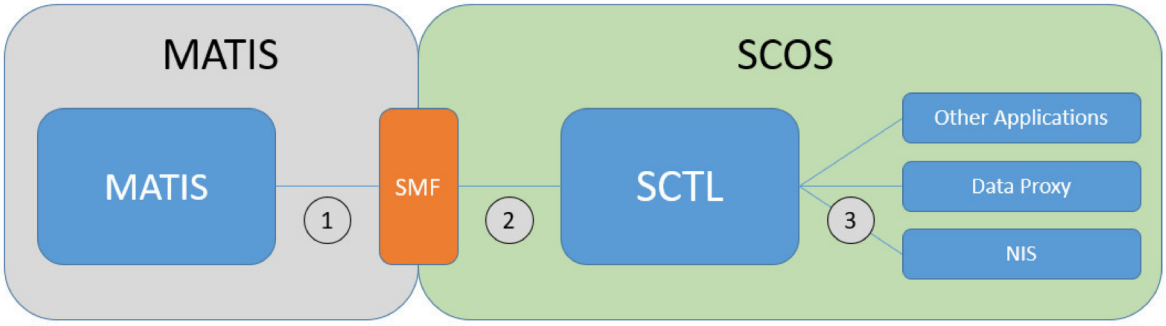


Figure 3.1: The connection between MATIS and applications in SCOS.

Note for SCOS users: The SCTL can be accessed via the EUD and opening the SCTL perspective. Additionally, the exact names for the applications and their settings can be taken from the TKMA file used by SCOS where applications are defined.

3.2.2 FMS

FMS is based on the programming language Groovy. Therefore, direct support within MATIS is available. This results in two possibilities. On the one hand, this Groovy

script can be placed on the OSMCs machines and be triggered by MATIS with a call of a script from MATIS. On the other hand, FMS can be implemented in MATIS and be triggered directly within a procedure. Currently, MATIS will trigger the FMS Groovy script on the OSMCs machine due to testing on these machines. Nevertheless, in the future it is intended to transfer this script to MATIS itself.

3.2.3 SpaceShell

To use SpaceShell with MATIS, two processes have been implemented. Most importantly, due to its complex terminal input, commands executing SpaceShell will be written into a file on the corresponding machine using the SwissKnife driver in MATIS. When executed, SpaceShell commands will be triggered. After execution, the script gets deleted to clean up the machine it was running on.

3.2.4 OSMATx

While testing the MATIS system a massive start-up time has been experienced. After investigation the resolution for the problem was to increase the processor speed and the RAM of the virtual machine. This was necessary since the system ran out of RAM due to the increasing number of procedures implemented and therefore, the processor tried to manage the insufficient RAM resulting in a full CPU load and very long start-ups of MATIS.
Later, a second enhancement has been done when a SCOS-client application was implemented and both systems were running at the same time. Thus, the virtual machine peripherals have now been quadrupled to ensure smooth mission operation automation execution.

3.2.5 CSP-term

CSP-term is used interactively, therefore it will be implemented with a special bash script. This bash script is executed then and the output is piped back into MATIS for the results. CSP-term is a live programme awaiting input from the user in the command line after it is started. This is not possible in automation, therefore a special bash syntax has been discussed but is yet to be implemented.

Chapter 4

Implementation and Converters

In the following section, the implementation of OPS-SAT's automation will be introduced. Additionally, in-depth details about the problems regarding changes from already existing manual operation and their procedures to autonomous execution will be given. This change from manual procedures executed by operation engineers in SCOS to the automatic execution in MATIS introduces a converter, which allows excel procedures to be converted into the PLUTO language. Besides the operation automation concept and automation integration, the converter is one of the key achievements in this master thesis. It will allow the verbose PLUTO procedures to be generated automatically. At the current number of procedures, automatic conversion saves the operator ten-thousands of lines of code to be written and an enormous amount of time to implement over **369** procedures from scratch.
The additional "Excel Procedures to MATIS System Element Configuration File Converter" (see Section 4.2) generates the "se.xml" file which is necessary in every system element folder to define the structure and configuration of the SE itself as well as the associated procedures. The converter handles the maximum execution duration of a procedure until timeout, input arguments of all procedures, its description and type and saves all information in an ".se.xml" file. Each System Element contains one .se.xml file containing the corresponding procedure configuration. MATIS uses these files to extract the mentioned features of a procedure and to generate the internal structure for the SEs in the MATIS system. The third and last converter introduced during this master thesis is the "SCOS to MATIS MISC MIB Converter". As the name suggests, this converter takes the MISC MIB from SCOS and converts it into a format usable for MATIS. Its necessity and meaning can be extracted from Section 4.3 below.

4.1 Excel Procedures to PLUTO Code Converter

This converter tackles the main issue of a transition from manual operation to automation. Due to already existing procedures, its implementation into an automation system can

take several months of coding and thousands of euros, which can be a project breaker for smaller missions.
An additional feature of this converter is the automatically generated folder structure for an easy drag and drop from the converter into the repository of MATIS. Furthermore, missing functionality in the PLUTO programming language is implemented by the converter itself, using workarounds. As an example, ranges, enumerations, lists and GOTO are not supported in PLUTO. Moreover, this converter calls a second converter responsible to generate the necessary .se.xml files which must be contained inside of each System Element repository folder to identify each procedure. This additional converter and its functionality are explained in further detail in Section 4.3.
Additionally, all code of the converters can be found in Appendices E, E and G.

4.1.1 Input

The input for the converter are the Excel procedures written by the operator. The folder with the Excel procedures has to be placed into the "EXCEL" folder inside the workspace.
The structure inside the EXCEL folder will automatically be contained for the generated PLUTO procedures. Therefore, the procedures are already well structured when coming out of the generator and can be dragged and dropped into the MATIS directory directly. The naming convention of the folders inside the EXCEL folder will be kept. This process is depicted in Figure 4.1.
To give operation engineers some freedom within their procedure repository, all folders named "old" will not be used as input for the converter. Therefore, operation engineers can use it as a backup folder. Nevertheless, it can also be used for storing procedures that should not be implemented into MATIS yet. In addition to this, all files starting with "~" are automatically ignored. This is a safety mechanism to prevent temporary files from being opened.

The naming convention is the following:

- Folders to be used shall not be named "old"
- Files shall not start with "~"
- Procedures shall start with a capital indication string (e.g. "LEOP", "R") followed by their category and their name.
- The Procedure name shall not contain "_"
- Procedure name and description shall be separated by "_"

- Any "-" in the procedure name are converted to "_" for correct name syntax in MATIS
- Procedure descriptions shall not include spaces. Spaces shall be replaced by "_"
- Characters after the procedure name will be formatted to the procedure description. Any "_" will be replaced by a space.
  (E.g. R-ADC-N310_Start_of_experiment_v8.xlsx becomes R_ADC_N310.PLUTO with description "Start of experiment v8")

4.1.2 Output

The output of the converter are the PLUTO procedures and the corresponding se.xml files. Additionally, the folder structure is inherited. Se.xml files are content files describing the PLUTO procedures inside of a folder. These files tell MATIS about the arguments with types and description of each PLUTO procedures. Each folder containing a minimum of one PLUTO procedure needs exactly one se.xml file.
As described in the section above, the names will be converted to a standardised string, starting with an identifier (e.g. "LEOP" or "R") followed by its name/ID and a description. The outputted structure is visualised in the illustration below.

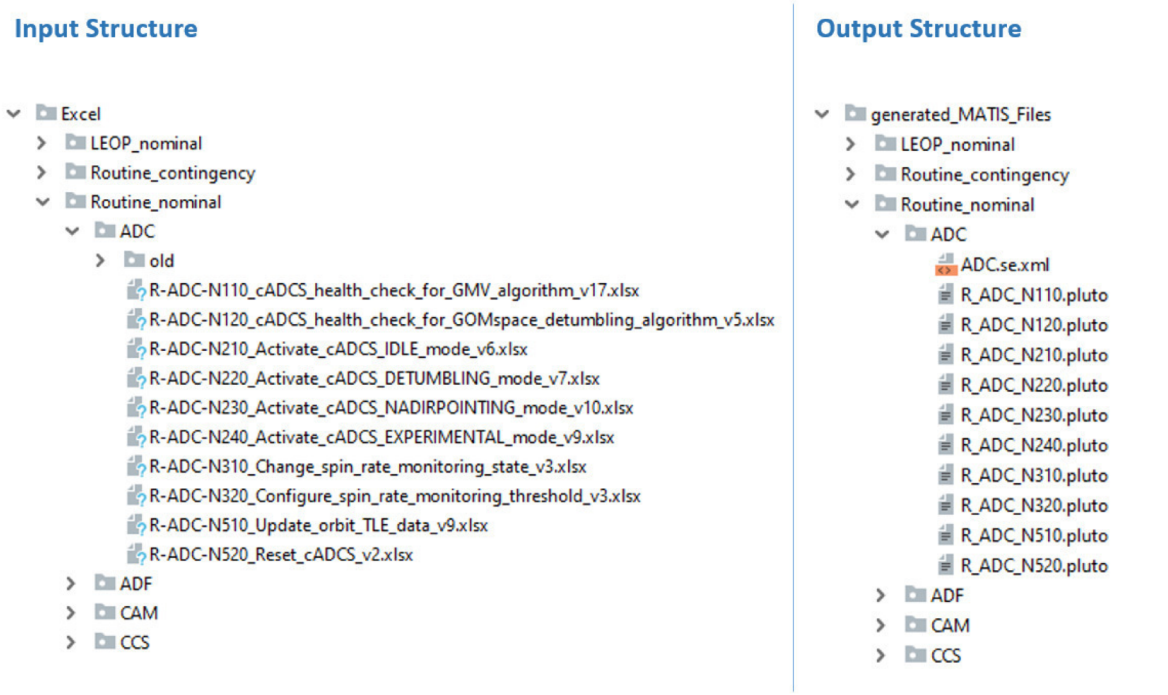


Figure 4.1: Structural comparison between input and output files and directory.

4.1.3 Standards and Conventions

Due to the static code of the converter, a new standard for Excel procedures has been developed in cooperation with the Operations Engineer.
In this section, the convention for Excel procedures to be converted to PLUTO procedures is defined. Additionally, some basic information about the corresponding functions in the Python code is given to provide easier access and adaptation with respect to the new functionality added to the Excel procedure standards for future operations.
The current design of the converter checks for operation keywords in the OPERATIONS column of the Excel procedure. Only valid operation keywords will trigger the functions inside of the converter and write the corresponding code as text inside of the PLUTO procedure file.
The keywords for the OPERATIONS column in the Excel procedure can be taken from _KNOWN_OPERATIONS_PARAMETER inside of the Python code of the converter. Additional operations have to be added inside of this variable before new functions can be added. It should finally be noted that these `_KNOWN_OPERATIONS_PARAMETER` are written in all-caps and without spaces.

Conventions for Dependencies

In the PLUTO language, it is important to include the dependencies, meaning the location inside of the MATIS repository, to a command, reporting data, schedule or procedure. In comparison to conventional file dependencies in other programming languages, PLUTO uses the word "of" as a dependency operator. The following example clarifies the approach.

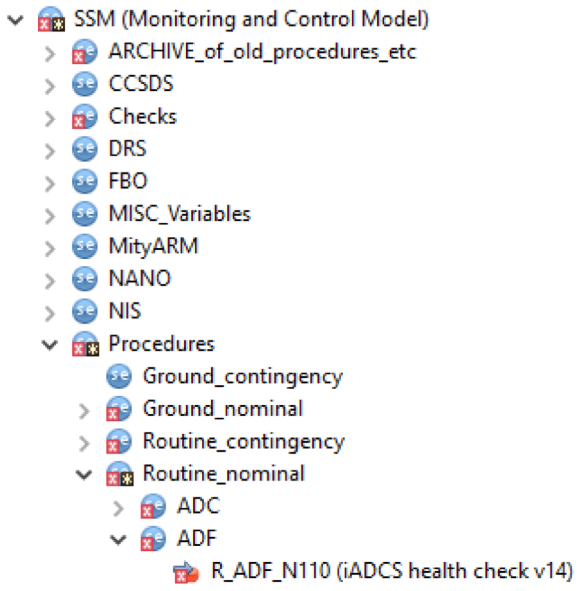


Figure 4.2: An example of the MATIS repository structure.

In most programming languages, the procedure "R_ADF_N110" is referred to as /SSM/Procedures/Routine_nominal/ADF/R_ADF_N110. In PLUTO, this same procedure is referred to as "R_ADF_N110 of ADF of Routine_nominal of Procedures of SSM".
The converter generates the dependencies for given activities automatically. Nevertheless, in the converter, these structures have to be defined inside of the Python functions `create_PROCEDURE_dictionary()` and `create_parameter_dictionary()` for procedure location and TM parameter/TC commands respectively.

Convention for Commands and Operations

Every command and operation has to be indicated by a corresponding text in the OPERATIONS column of the Excel procedure. Commands and operations can have an ID, a Description, Type, raw and engineering value, a unit, a display and the different confirmation stages of the execution in SCOS. Commands and operations have either one line without, or multiple lines with arguments. Additionally, there are two different kinds of additional arguments. One type of arguments are **arguments** of the specific command, so the command argument ID is entered directly below the command or operation. The other type of argument is called **directives**, such as EXECUTION TIME or DYNAMIC PTV. These directives do not have an ID but are written into the DESCRIPTION column below all other arguments of this command. This is an important differentiation and can be used for any kinds of commands. Nevertheless, arguments are statically allocated to a procedure and therefore, cannot change, whereas directives can be added to any arbitrary command.

Type Conversion

The Python function executing this operation is `convert_TYPE_from_SCOS_to_MATIS()` Different conventions for Excel and PLUTO types are used. The convention to adapt these values to PLUTO types can be seen in the enumeration below. The left hand side represents the Excel convention and the right hand side depicts the corresponding PLUTO type the original type is being converted to:

- Enum, U8, U16, U32, U62 → Unsigned integer
- S8, S16, S32, S64 → Signed integer
- Boolean → Boolean
- Float → Real
- Octet Str, Char Str → String

- Abs Time → Absolute time
- Del Time → Relative time

SEND AND CHECK TCV

This operation is accomplished by the Python functions `write_SEND_WITH_TCV()` and `write_SEND_WITH_TCV_()`.
This operation sends a command and waits for its execution confirmation before continuation. If a command was not successful or time runs out, the procedure will aborted. An example can be seen in Figures 4.3 and 4.4.

Without Arguments

The command SEND AND CHECK TCV without any additional arguments generates only the necessary parts of the command to send the command correctly (see orange and green colours in Figures 4.3 and 4.4.)

With Arguments

"With arguments", the arguments are iterated and added to the command in between the strings "with arguments" and "end with". The two Figures 4.3 and 4.4 below can be used for comparison, for clarification, the arguments block is marked in blue.

With Directives

"With directives" is used to give a command a certain configuration.
All valid configurations with their possible assignments are listed below:

- AUTHORISATION DISABLED
  This function gives the opportunity to set-up commands without sending them to the spacecraft yet. Commands with authorisation disabled wait until a command without this option is sent. Then all previously defined commands with option authorisation disabled are sent in their previously transferred order.
- EXECUTION VERIFICATION DISABLED
  Adding this option disables the execution verification. Therefore, the command labelled as confirmed immediately after it left the ground station successfully and ignores the possible execution verification coming from the spacecraft.
- DYNAMIC PTV OVERRIDE
  This option ignores the check for TC FLOW coming into SCOS and sends the command regardless of its status.
- STATIC PTV OVERRIDE
  This option ignores the check for TC FLOW and TM FLOW coming into SCOS and sends the command regardless of its status.

- RELEASE TIME
  The release time decides when a command is send from the ground station to the spacecraft within SCOS.
  Examples of the format can be found below:
  - 2019-07-10T15:02:18.000
  - NOW + 1s
  - $NOW + 4d11h13min4s
- EXECUTION TIME
  The execution time decides when a command is executed on the spacecraft.
  Examples of the format can be seen below:
  - 2019-07-10T15:02:18.000
  - NOW + 1s
  - $NOW + 4d11h13min4s

For RELEASE TIME and EXECUTION TIME "NOW" and "$NOW" in the RAW column in the Excel procedure will automatically be converted to "current_time()" and the defined variable "NOW", respectively in PLUTO. Additionally, time can be added by "+<value>" with s, min, h and d as indicators for seconds, minutes, hours and days respectively.
By the end of the command, a semicolon ";" is added to conclude and indicate the end of the command.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG	UNIT
5.1	SEND AND CHECK TCV	M040603b	MC enableGeneration				
		LEN1	Length field	U16	2		
		4060300	aggregationId_L	U32		EP82	
		4060301	enableGeneration	Boolean	1		
		4060300	aggregationId_L	U32		EP88	
		4060301	enableGeneration	Boolean	1		
			RELEASE TIME		$T2		
			DYNAMIC PTV OVERRIDE				
			STATIC PTV OVERRIDE				
			EXECUTION TIME		$NOW+3s		
			CEV DISABLE				

Figure 4.3: An example of SEND AND CHECK TC colour-divided Excel representation.

```
initiate and confirm M040603b of NanomindTCs of MIB_TCs of Telecommands of SSM       //MC enableGeneration
   with arguments
      raw value of LEN1 := 2,     //TYPE: Unsigned integer      //DESCRIPTION: Length field
      ~4060300 := "EP82",         //TYPE: Unsigned integer
      ~4060301 := "1",        //TYPE: Boolean       //DESCRIPTION: enableGeneration
      ~4060300 := "EP88",         //TYPE: Unsigned integer
      ~4060301 := "1"         //TYPE: Boolean       //DESCRIPTION: enableGeneration
   end with
   with directives
      release_time := T2,
      dynamic_ptv := "overridden",
      static_ptv := "overridden",
      execution_time := current_time()+3s,
      execution_verification := "disabled"
   end with;
```

Figure 4.4: An example of SEND AND CHECK TC colour-divided PLUTO representation.

SEND

This operation is done by the Python functions `write_SEND()` and `write_SEND_()`. SEND is almost identical to SEND AND CHECK TCV in Section 4.1.3. Nevertheless it applies two major differences.
First, the string "and confirm" after "initiate" is being left out. Secondly, "wait for 0.5s;" is added below the command to avoid a confusion of the order of the commands in the back-end of MATIS to SCOS during execution, which could be critical for mission operation.

CHECK TM

This operation is conducted by the Python function `write_CHECKTM()`.
In the case of CHECK TM, a few options can be examined and must be differentiated.

Value Allocation

If the engineering or the raw value (from now on called "value cell") starts with "@", an allocation takes place and the value of the TM parameter is allocated to the variable located behind "@". Additionally, if the variable changes its type during the procedure, the converter declares the variable in the beginning of the code and changes its name to "VAL_" with an ending depending on the type of the variable.
Definition of "VAL_" depending on the PLUTO variable type:

- Unsigned integer → VAL_UI
- Signed integer → VAL_SI
- Boolean → VAL_BOOL
- Real → VAL_REAL
- String → VAL_STR
- Absolute time → VAL_ABST
- Relative time → VAL_RELT

(Conversion to PLUTO variable type can be found in Section 4.1.3)
After the allocation of the parameter, the TM value is logged. A visualisation of the exact conversion can be seen in Figures 4.5 and 4.6.

Range Check

Range Check is triggered by "[" in value cell and indicates the allowed range. If the range is exceeded, a warning message is raised. Ranges are not supported by PLUTO.

Therefore, a workaround is implemented by comparing the value to two size-indicators. An example of this mechanism is shown in the Figures 4.5 and 4.6 below.

Enumeration Check

Enumeration Check is triggered by "{" in value cell and enumerates through every entry inside of the waved brackets. Missing iteration functionality in PLUTO is substituted by a multi-conditional if condition. Each entry in the if condition is divided by an "or" statement and serves as sufficient workaround.

Size Comparison

This functionality is triggered by comparison operation engineers, these can either be ">", ">=", "<" or "<=". It compares a TM parameter to the value listed behind the comparison operator. A warning is triggered if the TM parameter comparison returns as "FALSE".

TM Check Based on Variable or Argument Value

This is triggered when the value cell starts with a dollar sign "$". The dollar sign is removed and the variable or argument is inserted for comparison. If the TM parameter is unequal to the variable or argument, a warning is raised and logged in MATIS.

TM Check Based on Fixed Value

This is the most common case and will be triggered when none of the above methods apply. In this case, the if condition checks for the cell value and gives a warning whenever the condition is not met.

For visualisation and clarification an example of each possibility is depicted in Figures 4.5 and 4.6 below.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG
	CHECK TM	ADCS1095	EXAMPLE1	U8	[1,3]	
	CHECK TM	ADCS1096	EXAMPLE2	U8	{1,2,3,4}	
	CHECK TM	ADCS1097	EXAMPLE3	U8	>=4	
	CHECK TM	ADCS1098	EXAMPLE4	U8	$TEST_VAR	
	CHECK TM	ADCS1098	EXAMPLE4	U8	$@VAL	
	CHECK TM	ADCS1099	EXAMPLE5	U8	5	

Figure 4.5: An example of a CHECK TM Excel Procedure.

```
// DESCRIPTION: EXAMPLE1, ID: ADCS1095
if raw_value of ADCS1095 of ADC of MIB_TMs of Telemetry of SSM < 1 or ...
                raw_value of ADCS1095 of ADC of MIB_TMs of Telemetry of SSM > 3 then
    warn "LOG: FAILURE; ID: ADCS1095, TYPE: Unsigned integer, expected: [1,3], got: " + ...
                raw_value of ADCS1095 of ADC of MIB_TMs of Telemetry of SSM + ", DESCRIPTION: EXAMPLE1";
end if;

// DESCRIPTION: EXAMPLE2, ID: ADCS1096
if raw_value of ADCS1096 of ADC of MIB_TMs of Telemetry of SSM != 1 and ...
                raw_value of ADCS1096 of ADC of MIB_TMs of Telemetry of SSM != 2 and ...
                raw_value of ADCS1096 of ADC of MIB_TMs of Telemetry of SSM != 3 and ...
                raw_value of ADCS1096 of ADC of MIB_TMs of Telemetry of SSM != 4 then
    warn "LOG: FAILURE; ID: ADCS1096, TYPE: Unsigned integer, expected: 1,2,3,4, got: " + ...
                raw_value of ADCS1096 of ADC of MIB_TMs of Telemetry of SSM + ", DESCRIPTION: EXAMPLE2";
end if;

// DESCRIPTION: EXAMPLE3, ID: ADCS1097
if raw_value of ADCS1097 of ADC of MIB_TMs of Telemetry of SSM >=4 then
    warn "LOG: FAILURE; ID: ADCS1097, TYPE: Unsigned integer, expected: >=4, got: " + ...
                raw_value of ADCS1097 of ADC of MIB_TMs of Telemetry of SSM + ", DESCRIPTION: EXAMPLE3";
end if;

// DESCRIPTION: EXAMPLE4, ID: ADCS1098
if raw_value of ADCS1098 of ADC of MIB_TMs of Telemetry of SSM = TEST_VAR then
    warn "LOG: FAILURE; ID: ADCS1098, TYPE: Unsigned integer, expected:" + TEST_VAR + ...
                ", got: " + raw_value of ADCS1098 of ADC of MIB_TMs of Telemetry of SSM + ", DESCRIPTION: EXAMPLE4";
end if;

// DESCRIPTION: EXAMPLE4, ID: ADCS1098
VAL_UI := raw_value of ADCS1098 of ADC of MIB_TMs of Telemetry of SSM;
    log "LOG: CHECK TM ASSIGNMENT: VALUE = " + VAL_UI + "; DESCRIPTION: EXAMPLE4; ID: ADCS1098";

// DESCRIPTION: EXAMPLE5, ID: ADCS1099
if raw_value of ADCS1099 of ADC of MIB_TMs of Telemetry of SSM = 5 then
    warn "LOG: FAILURE; ID: ADCS1099, TYPE: Unsigned integer, expected: 5, got: " + ...
                raw_value of ADCS1099 of ADC of MIB_TMs of Telemetry of SSM + ", DESCRIPTION: EXAMPLE5";
end if;
```

Figure 4.6: An example of a CHECK TM PLUTO Procedure with manually cut and indented lines for visibility and clarity.

In the Figure 4.6 the three dots "..." represent a line cut. Additionally, they are used for visualisation only and do not represent PLUTO syntax. The code generated by the converter does neither have line breaks in between nor three dots. Therefore, lines connected with "..." represent one line in the PLUTO code generated by the converter.

DECLARE VARIABLES

This operation is done by the Python function `write_DECLARE_VARIABLES()`.
This function declares the variables present after OPERATIONS cell contains the keyword "DECLARE VARIABLES". It translates the type, name and description to the PLUTO language.
The parameter is then declared inside of the procedure as depicted in Figures 4.7 and 4.8.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG
1.3	DECLARE VARIABLES	$VAL	Generic variable for reading TM parameter values for which no specific value or range is expected	N/A		
		$SS_X	x component of measured sun vector	Float		
		$SS_Y	y component of measured sun vector	Float		
		$SS_Z	z component of measured sun vector	Float		
		$SUNVEC_X	x component of converted sun vector	Float		
		$SUNVEC_Y	y component of converted sun vector	Float		
		$SUNVEC_Z	z component of converted sun vector	Float		

Figure 4.7: An example of a DECLARE VARIABLES Excel Procedure.

```
declare
    variable SS_X of type Real described by "x component of measured sun vector",
    variable SS_Y of type Real described by "y component of measured sun vector",
    variable SS_Z of type Real described by "z component of measured sun vector",
    variable SUNVEC_X of type Real described by "x component of converted sun vector",
    variable SUNVEC_Y of type Real described by "y component of converted sun vector",
    variable SUNVEC_Z of type Real described by "z component of converted sun vector"
end declare
```

Figure 4.8: An example of a DECLARE VARIABLES PLUTO Procedure.

Declare undefined variables

The activity is accomplished by the Python function `write_DECLARE_TM_CHECK_VARIABLES()`.
An additional declaration of variables occurs in the beginning of the Python code. The script scans through the Excel file, searching for the character sequence "@$" in the RAW and ENG cells. When such a variable is found, its type is checked, its name is changed to VAL_<type> and declared in the PLUTO code beneath the results of `write_DECLARE_VARIABLES()`. A detailed description of the parameter types and its VAL_<type> conversion can be found in Section 4.1.3 in the paragraph "Value Allocation". Additionally, variables of the same type and name are not declared twice during this process. For the same procedure as shown in Figures 4.7 and 4.8, this additional feature looks as follows.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG
4.1	CHECK TM	ADCS1095	ADCS_MODE_VALUE	U8	@$VAL	
4.2	CHECK TM	CADC0875	I_COAR_NAD_FL	Boolean	@$VAL	
	CHECK TM	CADC0896	I_POS_FI_GPS_0	Float	@$VAL	
	CHECK TM	CADC0877	I_EXP_MONIT_FL	Boolean	@$VAL	
	CHECK TM	CADC0878	I_NAV_FLAG	Boolean	@$VAL	
	CHECK TM	CADC0879	I_NAV_TRANS_FL	Boolean	@$VAL	

Figure 4.9: An example of a DECLARE VARIABLES and undeclared variable declaration Excel Procedure.

```
declare
    variable SS_X of type Real described by "x component of measured sun vector",
    variable SS_Y of type Real described by "y component of measured sun vector",
    variable SS_Z of type Real described by "z component of measured sun vector",
    variable SUNVEC_X of type Real described by "x component of converted sun vector",
    variable SUNVEC_Y of type Real described by "y component of converted sun vector",
    variable SUNVEC_Z of type Real described by "z component of converted sun vector",

    variable VAL_UI of type Unsigned integer,
    variable VAL_BOOL of type Boolean,
    variable VAL_REAL of type Real
end declare
```

Figure 4.10: An example of a DECLARE VARIABLES and undeclared variable declaration PLUTO Procedure.

An example of all possible conversion is given in Figure 4.11 and 4.12.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG
	CHECK TM	ADCS1095	EXAMPLE1	U8	@$VAL	
	CHECK TM	CADC0876	EXAMPLE2	Boolean	@$VAL	
	CHECK TM	CADC0896	EXAMPLE3	Float	@$VAL	
	CHECK TM	CADC0878	EXAMPLE4	S16	@$VAL	
	CHECK TM	CADC0879	EXAMPLE5	Float	@$VAL	
	CHECK TM	CADC0880	EXAMPLE6	Enum	@$VAL	
	CHECK TM	CADC0881	EXAMPLE7	Octet Str	@$VAL	
	CHECK TM	CADC0883	EXAMPLE9	Del Time	@$VAL	
	CHECK TM	CADC0882	EXAMPLE8	Abs Time	@$VAL	

Figure 4.11: An example of a Type conversion Excel procedure.

```
declare
    variable VAL_UI of type Unsigned integer,
    variable VAL_BOOL of type Boolean,
    variable VAL_REAL of type Real,
    variable VAL_SI of type Signed integer,
    variable VAL_STR of type String,
    variable VAL_RELT of type Relative time,
    variable VAL_ABST of type Absolute time
end declare
```

Figure 4.12: An example of corresponding Type conversion PLUTO procedure.

SELECT CASE, CASE:, CASE ELSE, ENDCASE

This actions are done by the Python functions `write_SELECT_CASE()`, `write_CASE()`, `write_CASE_ELSE()` and `write_END_CASE()`.
These different functions are triggered by the corresponding string in the OPERATIONS column. The Python functions take care of the logic behind the case function inside of the Excel procedure and translate them adaptively into PLUTO. An example can be seen in the figures below.

STEP	OPERATIONS
	SELECT CASE $FACE
	CASE: CAMERA
	$PAYL_X := 0
	$PAYL_Y := 0
	$PAYL_Z := -1
	$FLIGHT_X := 0
	$FLIGHT_Y := 1
	$FLIGHT_Z := 0
	CASE: SBAND-PLUSX
	$PAYL_X := 1
	$PAYL_Y := 0
	$PAYL_Z := 0
	$FLIGHT_X := 0
	$FLIGHT_Y := 0
	$FLIGHT_Z := -1
	CASE: SBAND-MINUSX
	$PAYL_X := -1
	$PAYL_Y := 0
	$PAYL_Z := 0
	$FLIGHT_X := 0
	$FLIGHT_Y := 0
	$FLIGHT_Z := -1
	CASE ELSE:
	CALL ENGINEER
	Undefined pointing configuration
	END CASE

Figure 4.13: An example of a SELECT CASE Excel Procedure.

```
in case FACE
    is = "CAMERA":
        PAYL_X := 0;
        PAYL_Y := 0;
        PAYL_Z := -1;
        FLIGHT_X := 1;
        FLIGHT_Y := 0;
        FLIGHT_Z := 0;
    or is = "SBAND-PLUSX":
        PAYL_X := 0;
        PAYL_Y := 1;
        PAYL_Z := 0;
        FLIGHT_X := 0;
        FLIGHT_Y := 0;
        FLIGHT_Z := -1;
    or is = "SBAND-MINUSX":
        PAYL_X := 0;
        PAYL_Y := -1;
        PAYL_Z := 0;
        FLIGHT_X := 0;
        FLIGHT_Y := 0;
        FLIGHT_Z := -1;
    otherwise:
        initiate and confirm Send of Email of Communicator of SwissKnife of PRIME of D0 of TEST_MISSION of SMF
            with arguments
                Subject := "MATIS: CALL ENGINEER",
                Message := "      Undefined pointing configuration",
                ToMail := "Felix.Hessinger@gmail.com",
                ToName := "Operators"
            end with;
end case;
```

Figure 4.14: An example of a SELECT CASE converted PLUTO procedure.

IF, ELSE IF, THEN, ELSE, END IF, IF IN, ELSE IF IN

Regarding IF conditions two cases have to be distinguished. These are explicitly explained in the following sub-sections.

IF with Normal Condition

The operations are done by the Python functions `write_IF()`, `write_ELSEIF()`, `write_THEN()`, `write_ELSE()` and `write_ENDIF()`.

This iteration searches for the condition of the IF function and converts its context and condition into PLUTO procedure code. An exact implementation is displayed below.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG
	IF VAL_UI = 1 THEN					
	CHECK TM	ADCS1095	EXAMPLE2	U8	4	
	ELSEIF VAL_UI = 2 THEN					
	CHECK TM	ADCS1095	EXAMPLE4	U8	5	
	ELSE					
	CHECK TM	ADCS1095	EXAMPLE5	U8	6	
	END IF					

Figure 4.15: An example of an IF Excel Procedure.

```
if VAL_UI = 1 then
    // DESCRIPTION: EXAMPLE2, ID: ADCS1095
    if raw_value of ADCS1095 of ADC of MIB_TMs of Telemetry of SSM = 4 then
        warn "LOG: FAILURE; ID: ADCS1095, TYPE: Unsigned integer, expected: 4, got: " + ...
                raw_value of ADCS1095 of ADC of MIB_TMs of Telemetry of SSM + ", DESCRIPTION: EXAMPLE2";
    end if;

else if VAL_UI = 2 then
    // DESCRIPTION: EXAMPLE4, ID: ADCS1095
    if raw_value of ADCS1095 of ADC of MIB_TMs of Telemetry of SSM = 5 then
        warn "LOG: FAILURE; ID: ADCS1095, TYPE: Unsigned integer, expected: 5, got: " + ...
                raw_value of ADCS1095 of ADC of MIB_TMs of Telemetry of SSM + ", DESCRIPTION: EXAMPLE4";
    end if;

else
    // DESCRIPTION: EXAMPLE5, ID: ADCS1095
    if raw_value of ADCS1095 of ADC of MIB_TMs of Telemetry of SSM = 6 then
        warn "LOG: FAILURE; ID: ADCS1095, TYPE: Unsigned integer, expected: 6, got: " + ...
                raw_value of ADCS1095 of ADC of MIB_TMs of Telemetry of SSM + ", DESCRIPTION: EXAMPLE5";
    end if;
end if;
end if;
```

Figure 4.16: An example of an IF converted PLUTO procedure.

IF with Enumeration

The operations are done by the Python functions `write_IFIN()`, `write_ELSEIFIN()`. Enumerations are not supported by PLUTO language and have to be implemented separately by the usage of "or"-conditions. An example can be seen below.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG
	IF $PARAM IN					
		CAM2672p	BS_EXP16_CFG	U8		
		CAM2692p	BS_EXP16_OUT	U8		
		CAM2712p	BS_PWRSW_ALRT	U8		
		CAM2722p	BS_PWRSW_CTL	U8		
		CAM2732p	BS_PWRSW_FAULT	U8		
		CAM2752p	BS_PWRSW_SENSE	U8		
		CAM2762p	BS_PWRSW_SOURCE	U8		
		CAM2772p	BS_PWRSW_STS	U8		
		CAM2792p	BS_PWRSW_ALRT	U8		
		CAM2802p	BS_PWRSW_CTL	U8		
		CAM2812p	BS_PWRSW_FAULT	U8		
		CAM2832p	BS_PWRSW_SENSE	U8		
		CAM2842p	BS_PWRSW_SOURCE	U8		
		CAM2852p	BS_PWRSW_STS	U8		
		CAM2872p	RW_BS_PWRSW_ALRT	U8		
		CAM2882p	RW_BS_PWRSW_CTL	U8		
		CAM2892p	RW_BS_PWRSW_FAULT	U8		
		CAM2912p	RW_BS_PWRSW_SENSE	U8		
		CAM2922p	RW_BS_PWRSW_SOURCE	U8		
		CAM2932p	RW_BS_PWRSW_STS	U8		
		CAM2992p	PL_TS_CONFIG	U8		
	THEN $TYPE := "U8"					

Figure 4.17: An example of an IF IN Excel Procedure.

```
if PARAM = "CAM2672p" or PARAM = "CAM2692p" or PARAM = "CAM2712p" or PARAM = "CAM2722p" or ...
            PARAM = "CAM2732p" or PARAM = "CAM2752p" or PARAM = "CAM2762p" or PARAM = "CAM2772p" or ...
            PARAM = "CAM2792p" or PARAM = "CAM2802p" or PARAM = "CAM2812p" or PARAM = "CAM2832p" or ...
            PARAM = "CAM2842p" or PARAM = "CAM2852p" or PARAM = "CAM2872p" or PARAM = "CAM2882p" or ...
            PARAM = "CAM2892p" or PARAM = "CAM2912p" or PARAM = "CAM2922p" or PARAM = "CAM2932p" or ...
            PARAM = "CAM2992p" then
        ~TYPE := "U8";
end if;
```

Figure 4.18: An example of an IF IN converted PLUTO procedure.

DOLLAR SIGN

The operations are performed by the Python function `write_DOLLAR()`.
An OPERATIONS cell starting with a "$" sign is an indicator for a variable allocation. Therefore, the variable will be assigned to a value. The syntax conversion from Excel to PLUTO is done automatically.

CALL PROCEDURE, THEN RETURN

The functionality is achieved by the Python function `write_CALLPROCEDURE()`.
The converter is triggered by the keyword "CALL PROCEDURE". Using this makes it possible to call a new procedure within a procedure. The converter converts the syntax for the usage of the MATIS function to call a procedure. The procedure execution enters the new sub-procedure. After the sub-procedure is executed, the original procedure continues.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG
3.1	CALL PROCEDURE					
	ID: R-SYS-N350					
	TITLE: Change FDIR configuration for specific parameter					
	REASON: Call generic procedure					
	PARAMS:					
	R-SYS-N350.SUBSYS = "HDCamera"					
	R-SYS-N350.PARAM = $PARAM					
	R-SYS-N350.TYPE = $PTYPE					
	R-SYS-N350.LMIN = $LMIN					
	R-SYS-N350.LMAX = $LMAX					
	R-SYS-N350.WMIN = $WMIN					
	R-SYS-N350.WMAX = $WMAX					
	R-SYS-N350.ACTION = $ACTION					
	R-SYS-N350.CHECK = $CHECK					
	THEN RETURN					

Figure 4.19: An example of a CALL PROCEDURE Excel Procedure.

```
// CALL PROCEDURE:   ID: R-SYS-N350
// TITLE:  Change FDIR configuration for specific parameter
// REASON:  Call generic procedure
initiate and confirm R_SYS_N350 of SYS of Routine_nominal of Procedures of SSM
    with arguments
        SUBSYS := "HDCamera" ,
        PARAM  := PARAM,
        PTYPE  := PTYPE,
        LMIN   := LMIN,
        LMAX   := LMAX,
        WMIN   := WMIN,
        WMAX   := WMAX,
        ACTION := ACTION,
        CHECK  := CHECK
    end with;
```

Figure 4.20: An example of a CALL PROCEDURE PLUTO Procedure.

EXECUTE IN TERMINAL ON MCS MACHINE

The activities are done by the Python function `write_EXECUTEINTERMINALONMCSMACHINE()`.
This converter is triggered by the keyword "EXECUTE IN TERMINAL ON MCS MACHINE". This function converts a terminal command into a command executable by MATIS while accessing one of the OSMCs machines. Additionally, when a SpaceShell command is executed, the APID and Data Proxy port is automatically changed. This is necessary due to different "Users" like MATIS, operation engineers or experimenters, as they have different ports and APIDs to connect to. This differentiation is needed since experimenters do not have root access and signals from MATIS are routed differently.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG
	EXECUTE IN TERMINAL ON MCS MACHINE		$ dd if=/dev/zero of=~/test_SpW count=152 bs=1024			

Figure 4.21: An example of an EXECUTE IN TERMINAL ON MCS MACHINE Excel Procedure.

```
initiate and confirm execute_and_get_return of Command_Line of SSM
            with arguments
                COMMAND := " dd if=/dev/zero of=~/test_CAN count=150 bs=1024"
            end with;
```

Figure 4.22: An example of an EXECUTE IN TERMINAL ON MCS MACHINE PLUTO Procedure.

An example of a changed APID and port cannot be shown due to security reasons.

CALL ENGINEER

The operations are executed by the Python function `write_CALLENGINEER()`.
CALL ENGINEER triggers a function to send an email to all the operation engineers and corresponding people assigned to OPS-SAT via the SwissKnife driver of MATIS. Additionally, the running procedure in MATIS is paused and will wait for manual user input to continue with the procedure.
Operational note: The procedure should be stopped, the satellite manually checked and the schedule restarted again. Nevertheless, this process is optional and its execution can be decided by the operator.

WAIT

The logic is accomplished by the Python function `write_WAIT()`.
The keyword triggering this command is "wait for <RELATIVE_TIME>". The function translates a waiting time syntax and pauses the procedure during execution time for a certain amount of time. The time format has to be the relative time format as seen in the example in Section 4.1.3.

NOTE, CAUTION

The operations are conducted by the Python function `write_NOTE_CAUTION()`.
NOTE, CAUTION is triggered by the keywords "NOTE" and "CAUTION" in the OPERATIONS cell. This function writes its content as a comment into the PLUTO procedure code.

Cleaning-Process

After the converter was executed and the file has been written, a control protocol is executed and checks for non-ASCII characters and empty steps inside a procedure. Non-ASCII files are automatically recognised and deleted. The same is done when an empty step is found, erasing the step.

4.1.4 Limitations

Due to the complexity of mission operations, it is nearly impossible to implement a fully automatic conversion from the manual excel procedures which can be executed live with the supervision of an operator. Therefore, in some cases, the automatically converted PLUTO procedures have to be modified by hand. This section introduces some of the limitations.

Live Interaction of an Operator
Due to the possibility of complex live interactions during a supervised pass, some rare cases exist in which a conversion is not possible or implemented. In these cases, the command is written as a comment into the PLUTO procedure and has to be implemented manually, if possible.

4.2 Excel Procedures to MATIS System Element Configuration File Converter

The se.xml file generator is integrated into the Excel procedure to System Element XML converter and is executed after the PLUTO procedure generation.
As mentioned in Section 2.1.1, the converter generates se.xml files which contain information about the procedures and their configuration included inside of a folder with additional information about the input arguments, their description and type.
The se.xml files are generated from the "Parameter:" content of the Excel procedures. It includes all parameters listed inside of the Excel procedure and implements these inside of the se.xml file. Every procedure in a folder has its own section inside of the se.xml file.
An example for the input format and the output format for when only one procedure with arguments is in the current repository can be seen in Figure 4.23 and 4.24 respectively.

STEP	OPERATIONS	ID	DESCRIPTION	TYPE	RAW	ENG	UNIT	DISPLAY	R	GTO	A	A	STAGE_EX	A	E	C	CHECK
1	PREPARATION																
1.1	Stack: R-EXP-N310																
	TM/TC chain: any																
	Suitable for TTQ: yes (with modifications)																
	Parameters:	$EXPID	Experiment ID	U16													
		$FPGA	Does the experiment use the SEPP FPGA?	Boolean													
		$IADCS	iADCS usage (0=not needed, 1=access without power control, 3=pointing mode requested)	U8													
		$CAMERA	Camera usage (0=not needed, 1=access without power control, 2=access and power control)	U8													
		$ORX	Optical receiver usage (0=not needed, 1=access without power control, 2=access and power control)	U8													
		$SDR	SDR usage (0=not needed, 1=access without power control, 2=access and power control)	U8													
		$CCSDSE	CCSDS engine usage (0=no access needed, 1=access without power control)	U8													
		$GPS	GPS receiver usage (0=no access needed, 1=access without power control)	U8													
		$SBAND	S-band radio usage (0=no live pass needed, 3=pass requested)	U8													
		$XBAND	X-band transmitter usage (0=no pass needed, 3=pass requested)	U8													
		$NMF	Is this a NMF experiment (=1) or a non-NMF experiment (=0)?	Boolean													

Figure 4.23: Parameter declaration example inside of Excel procedure R_EXP_N310.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <LocalSystemElement SchemaVersion="1.0">
  - <SEObject Name="R_EXP_N310">
    - <ActivityDefinition ValidationState="draft" EstimatedDuration="000:00:05:00.000" Postconditions="" Preconditions="" Objectives="" Constraints="" Description="Start of experiment v8">
      - <Argument Name="EXPID" Description="Experiment ID">
          <Scalar Type="unsignedInteger"/>
        </Argument>
      - <Argument Name="FPGA" Description="Does the experiment use the SEPP FPGA?">
          <Scalar Type="boolean"/>
        </Argument>
      - <Argument Name="IADCS" Description="iADCS usage (0=not needed, 1=access without power control, 3=pointing mode requested)">
          <Scalar Type="unsignedInteger"/>
        </Argument>
      - <Argument Name="CAMERA" Description="Camera usage (0=not needed, 1=access without power control, 2=access and power control)">
          <Scalar Type="unsignedInteger"/>
        </Argument>
      - <Argument Name="ORX" Description="Optical receiver usage (0=not needed, 1=access without power control, 2=access and power control)">
          <Scalar Type="unsignedInteger"/>
        </Argument>
      - <Argument Name="SDR" Description="SDR usage (0=not needed, 1=access without power control, 2=access and power control)">
          <Scalar Type="unsignedInteger"/>
        </Argument>
      - <Argument Name="CCSDSE" Description="CCSDS engine usage (0=no access needed, 1=access without power control)">
          <Scalar Type="unsignedInteger"/>
        </Argument>
      - <Argument Name="GPS" Description="GPS receiver usage (0=no access needed, 1=access without power control)">
          <Scalar Type="unsignedInteger"/>
        </Argument>
      - <Argument Name="SBAND" Description="S-band radio usage (0=no live pass needed, 3=pass requested) ">
          <Scalar Type="unsignedInteger"/>
        </Argument>
      - <Argument Name="XBAND" Description="X-band transmitter usage (0=no pass needed, 3=pass requested) ">
          <Scalar Type="unsignedInteger"/>
        </Argument>
      - <Argument Name="NMF" Description="Is this a NMF experiment (=1) or a non-NMF experiment (=0)?">
          <Scalar Type="boolean"/>
        </Argument>
      </ActivityDefinition>
    </SEObject>
  </LocalSystemElement>
```

Figure 4.24: Example of a generated se.xml file for procedure R_EXP_N310 with only one Excel procedure in the current folder.

Conventions are the following:

The parameter declaration operator to trigger this function shall be "Parameters:" in the OPERATIONS column in the Excel procedure.

The first parameter declaration shall start in the same row as "Parameters:" appears in the Excel procedure. Additionally, an ID shall always start with "$" and contain the full name behind the "$".

Furthermore, there shall be no empty ID rows during parameter declaration enumeration and each paramter shall have a type. Lastly, the parameter type must contain one of the following SCOS types:

- Enum, U8, U16, U32, U64
- S8, S16, S32, S64
- Boolean
- Float
- Octet Str, Char Str
- Abs Time
- Del Time

4.3 SCOS to MATIS MISC MIB Converter

Space missions at ESA have a Mission Information Base (MIB) which provides the space mission with information about TM and TC parameters occasionally. This is crucial for the mission control system (here SCOS) and other programmes for being able to interpret parameters going to and coming from the spacecraft. A variety of files is provided, with each of them having its own category, this collection represents the file-pool of the MIB.
Of course, MATIS also needs this information to process and categorise the parameters and commands correctly. Nevertheless, MATIS uses an other format for its input file for MISC variables than SCOS does. Therefore, a converter has been developed to implement the MISCconfig.dat file in the right format.

Formats:
The format MATIS needs for its MISC variables inside of the MISCconfig.dat file is straight forward. Each line represents one variable. The first word in a line is the variable name. A description of this parameter can be implemented by adding a TAB and the description behind the TAB. This description shall not contain line breaks or non ASCII characters.
In comparison to this rather simple concept, the concept of the MIB for SCOS and its MISCcontext.dyn file is defined in its own ICD, and will not be further explained in this thesis due to complexity, but all information can be found in [31].
The Python code for the "SCOS to MATIS MISC MIB Converter" is attached in Appendix G.

4.4 Converter Execution

The converters can be executed using the command line on a Linux or Windows machine or by using an IDE like PyCharm.
For their execution Python 3.6 or newer must be installed on the system. Additionally, the packages described in the beginning of the Python code of each individual programme must be installed with its correct version number, if explicitly mentioned. Before execution, the files and structures to be converted must be placed into the Python workspace. The converters' output will appear in the workspace after the converter programme is executed.

Chapter 5

Automation

Automation in this document describes the machinable execution of tasks given by a Mission Planning System or an operator for a pass outside of the working hours.
The automation concept is kept as simple as possible. MATIS provides a fixed platform with all functionality and tested procedures necessary to execute every expected task given to it. Therefore, the automation consists out of two key elements. The first element is a static platform. It includes well tested procedures, activities and Groovy functions to be executed.
The second part are the Event Schedules (MAES) and User Schedules (MAUS) as an input from the Mission Planning. These files are automatically stored in a dedicated folder in MATIS called IN-TRAY. When a MAES is transferred into the IN-TRAY, it is automatically loaded into the MATIS Calendar with its time, ID and name. After the MAES is executed, the MAUS will be loaded into the Calendar as well and synchronises with the existing events. Thus, the tasks of the MAUS have a dedicated starting time relative to Events. The MAUS is then waiting for the right events and pre-conditions to be satisfied to execute its task. Tasks of the MAUS can execute procedures and activities.
To summarise, each procedure used for automation is well tested and static, therefore normally does not change. This leads to a well-tested system able to be used as a flexible tool for operation due to the variable stackability of tasks which contain procedures in a MAUS. The exact high level system is illustrated in Figure 5.1.

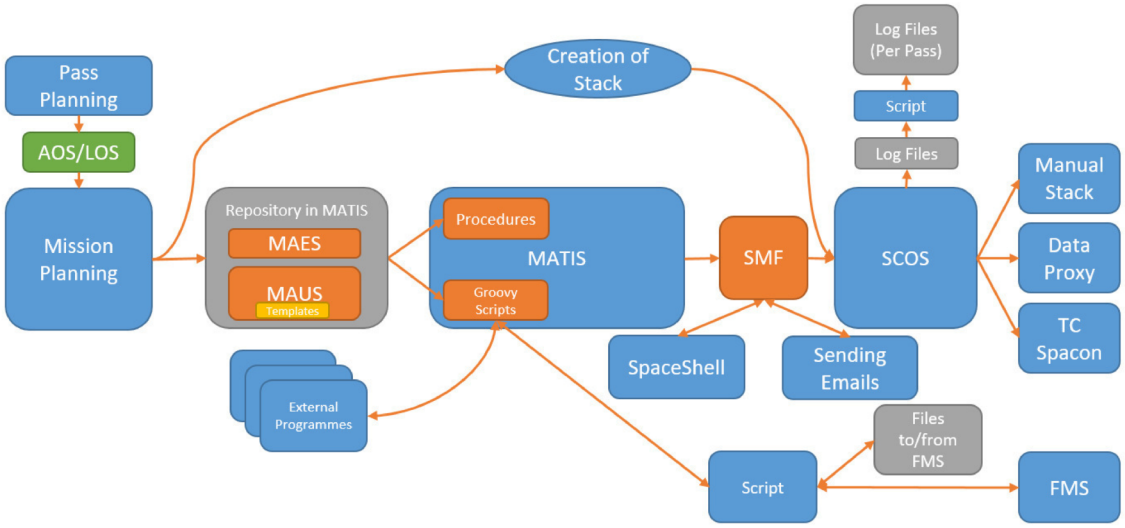


Figure 5.1: Input-output flow chart concerning MATIS.

It is worth to note that customised non-standard procedures to fulfil the needs of a certain situation can always be added during working hours but should be tested on a Flatsat prior to execution on the spacecraft.
In the current state of the Mission Planning System the MAES and MAUS preparation are done manually with little automation. Nevertheless, a fully automatic MPS will be developed and run during OPS-SAT's operation in space.
To ensure smooth integration of automation even without the currently missing Mission Planning Software, the system is designed in a way that automation preparation is able to be implemented by either operations engineers or by the MPS. This design choice eliminates the risk of the MPS not being developed until launch and the corresponding automation loss. In addition to eliminating this risk, this adaptive approach to work with both systems, the MPS and manual planning, has been chosen due to the current lack of a Mission Planning System in OPS-SAT. Nevertheless, the MPS will exist during operational use of OPS-SAT, eventually. This makes the adaptability of the automation system a necessity.

5.1 Concept

The OPS-SAT mission is highly flexible and therefore, the system has been adapted in order to keep up with its many possibilities during a pass.
In OPS-SAT operations, different passes have to be differentiated. Due to OPS-SAT's possibility to communicate either via UHF, S-Band or X-Band, different ground station and spacecraft configurations have to be changed accordingly. Without a synchronised

ground and spacecraft configuration, communication and commanding is not possible. As an example, if the ground station is set-up for UHF and the spacecraft is sending in S-Band, the ground station will not be able to receive and process the data sent from the spacecraft, resulting into data loss. Data loss in a crucial moment can result in critical situations.
In order to keep operations and preparation for a pass simple, different MAUSs are available. For every used telemetry combination, a MAUS is prepared. Therefore, preparing for a pass can easily be done by the MPS. The MPS chooses the right pass and transfers the corresponding MAUS into the IN-TRAY folder where it is automatically loaded into the MATIS calendar is illustrated in Figure 5.2.

Figure 5.2: An illustration of the IN-TRAY concept.

Each "Pass-Preparation"-MAUS includes tasks containing procedures to configure the ground segment that includes SCOS, Data Proxy, antenna, amplifier and signal lock settings. Additionally, on the spacecraft side of the MAUS, it contains commands to switch to the correct transmitter, receiver and internal communication mode to send telemetry data instead of saving it in the dedicated on-board storage (Packet Store). This concept takes the rather complex planning away from the MPS. Now the MPS decides on the type of pass and the MAUS takes care of the rest. This is an especially big advantage for manual mission planning since the commands do not have to be prepared one by one, reducing the human error and operational costs as far as possible. The same concept is extended to standard operations. Therefore, commands for downloading the saved telemetry from an orbit are prepared in a MAUS, as well as downloading experimenter data, if all telemetry and power constrains for these downloads are fulfilled.
Additionally, a MAUS for the end of a pass is prepared and can be executed shortly before the end of a pass to set all conditions for a save orbit. This includes deactivation of the spacecraft transmitter and configuration of the ground station systems to stand-by states, awaiting commands for the next pass.

Other steps are handled by the MPS itself, writing its own individual MAUS for each pass. As mentioned in Section 2.1.7, these decisions are based on the power demands and requirements of the experiment. An illustration of this concept is depicted in Figure 5.3.

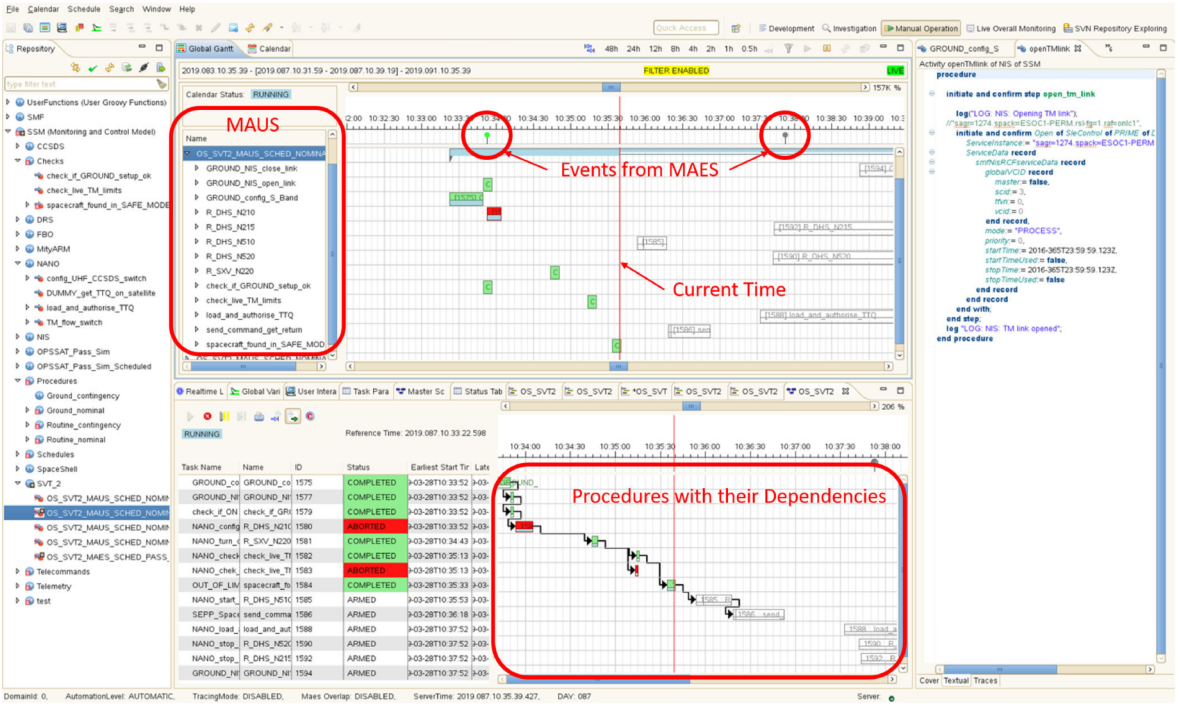


Figure 5.3: A screen capture which gives an example of the Operation Manager showing the Calendar with Events, MAUS, procedures and their dependencies.

5.2 Pass Concept

OPS-SAT's high complexity and configurability also sets challenges for its passes. The satellite is capable of communicating in UHF, S-Band and X-Band. Therefore, different ground and spacecraft configuration setups have to be considered. For nominal passes, four uplink and downlink configurations have to be investigated, which are depicted in the following Table 5.1.

Table 5.1: Communication mode for passes.

Mode	Uplink	Downlink
1	UHF	UHF
2	S-Band	S-Band
3	S-Band	X-Band
4	UHF	UHF X-Band

The typical case for its use is option two with an up- and downlink in S-Band. This

allows a relatively fast data transfer from and to the spacecraft. In this configuration, interactive application can be started from the ground. Among others, such applications are SpaceShell and FMS. Due to these programmes playing a fundamental role in OPS-SAT's operations, communication mode number two will be used most commonly. Changing up- and downlink to UHF provides the possibility to use CSP-term, accessing low level system components directly from the NanoCom and routing the command to corresponding peripherals as mentioned in Section 2.1.8.

X-Band can be used to downlink big data packages to the ground. Nevertheless, this mode is usually not being used operationally. The planned case for using X-Band passes are experiments that would like to downlink data via X-Band to the ground. In contrast to UHF and S-Band, X-Band is not processed in real time, but saved into a file which can be read at a later point in time.

This limitation is a key difference between X-Band and UHF or S-Band. Therefore, two options for the use of X-Band downlink result. Mode three offers a fast uplink in S-Band, allowing larger files to be transferred to the spacecraft, while X-Band downlinks data to the ground. This option does currently not allow feedback of successful execution of a command in real-time. This makes operations more difficult, since the operator or automation system has to trust that all commands are executed correctly on the spacecraft, leading to proper behaviour of the satellite.

To avoid the blind uplink of commands, mode four allows a UHF link between the satellite and ground and the other way around at the same time as the X-Band downlink. This renders the possibility of commands being sent with confirmation from the spacecraft, allowing operation engineers and the automation system to react according to the state of the command. This option allows fast downlink and both way communication at the same time but does not provide fast file uplink. For a better understanding of the actions during a pass, a more detailed concept for UHF and S-Band is described in Section 5.2.1 below.

5.2.1 Pass Event Timeline

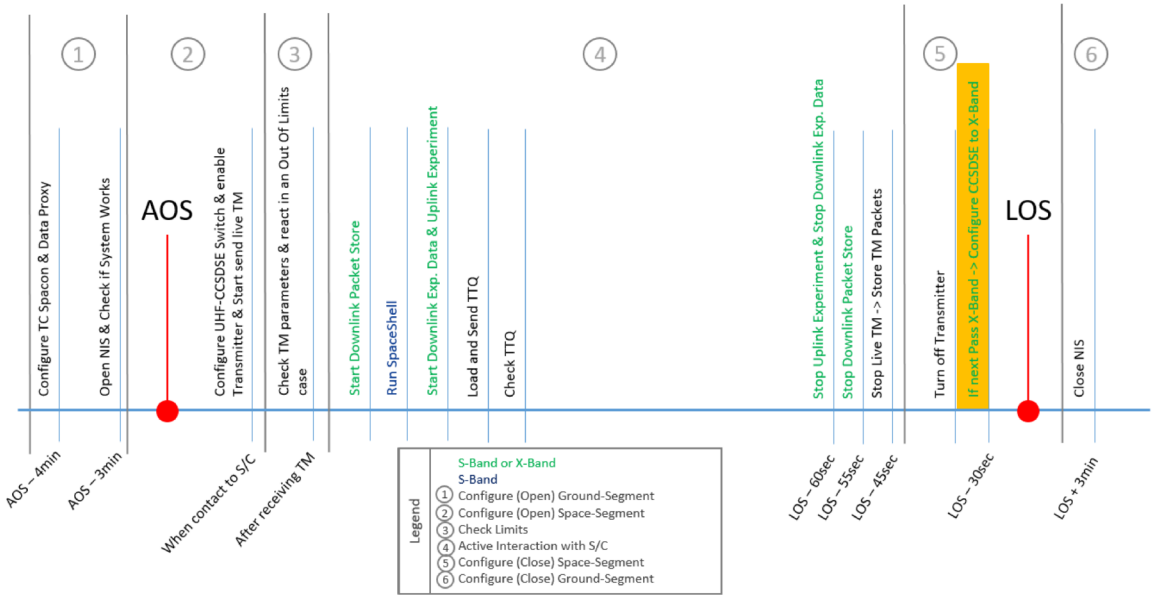


Figure 5.4: An example of the Timeline of a pass for different communication settings.

Figure 5.4 presents a rough timeline of events occurring during a pass. As mentioned before, its events are based on the up- and downlink configuration indicated in blue and green. The pass itself if divided into six phases.
The phases of a satellite pass for OPS-SAT are (i) "Configure (Open) Ground-Segment", (ii) "Configure (Open) Space-Segment", (iii) "Check Limits", (iv)"Active Interaction with S/C", (v)"Configure (Close) Space-Segment" and (vi)"Configure (Close) Ground-Segment". Each phase can be divided into individual events.

(i) Configure (Open) Ground-Segment
During this phase, the ground segment is being prepared for the upcoming pass. Preparation is done before contact with the spacecraft is established and finishes when a link between ground and satellite has been established. An overview of this phase is presented in Figure 5.5. Depending on the used frequency band, the signal is routed differently through the system and its hardware. Therefore, UHF usage requires the data proxy to set its path to "UBBE", whereas S-Band requires the configuration "CORTEX". This routes the signal to the transmitter for either UHF or S-Band. Additionally, the TC SPACON is configured to Packet Mode or CLTU Mode for UHF or S-Band respectively. This enables correct en- and decoding of the data, since UHF and S-Band have different standards in OPS-SAT.

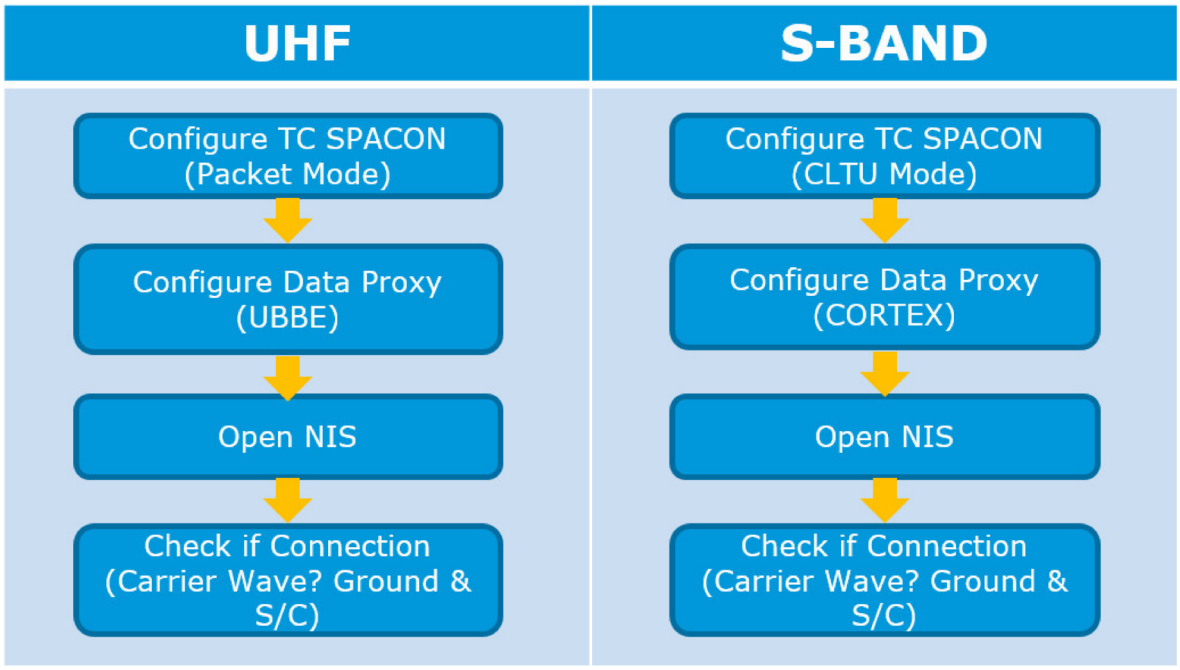


Figure 5.5: Detailed pass events for phase "Configure (Open) Ground-Segment".

(ii) Configure (Open) Space-Segment

After the ground peripherals are set up correctly, the spacecraft is configured. In Figure 5.6, the details for the transmitter on the spacecraft are shown. After successful TM channel switch, the TM flow is activated, no longer saving TM in the Packet Store but sending these to ground instead.

Caution has to be giving to the fact that data sent to the ground cannot be saved on-board simultaneously. Therefore, it is important to have locked on to the carrier wave of the spacecraft before activating TM flow on the satellite side.

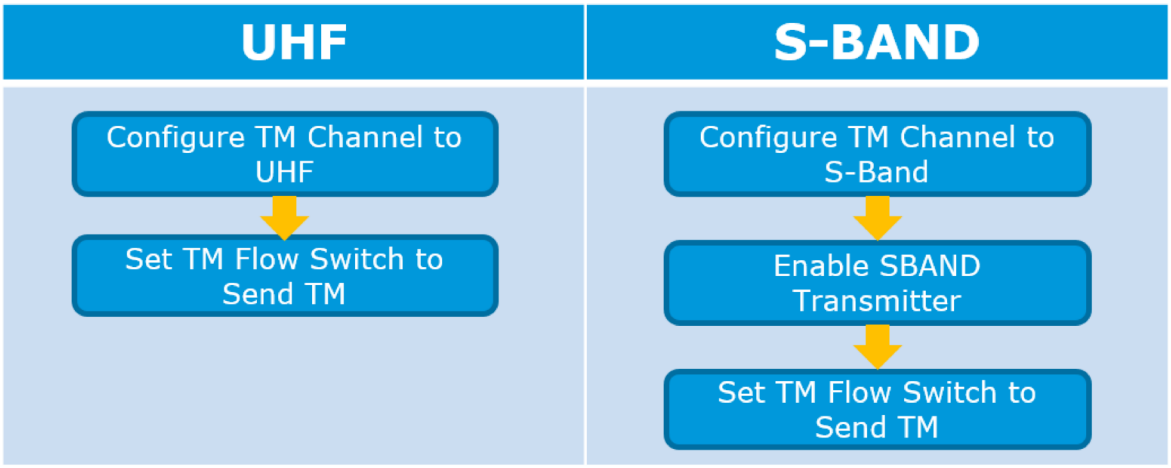


Figure 5.6: Detailed pass events for phase "Configure (Open) Space-Segment".

(iii) Check Limits

Checking limits and health of the spacecraft is an essential part during mission operation. It detects errors, unforeseen events happening during the last pass and helps to prevent critical situations in the first place.

When an anomaly is found, MATIS currently reacts very strictly and aborts all operations connected to the phase "Active Interaction with S/C" and sends an email with information about the anomaly to the operation engineers. This operation can be seen in Figure 5.7.

Nevertheless, this will be changed in future iterations to a more intelligent system, which aborts just the current procedure and executes special procedures to dedicated procedures which execute a pass safely.

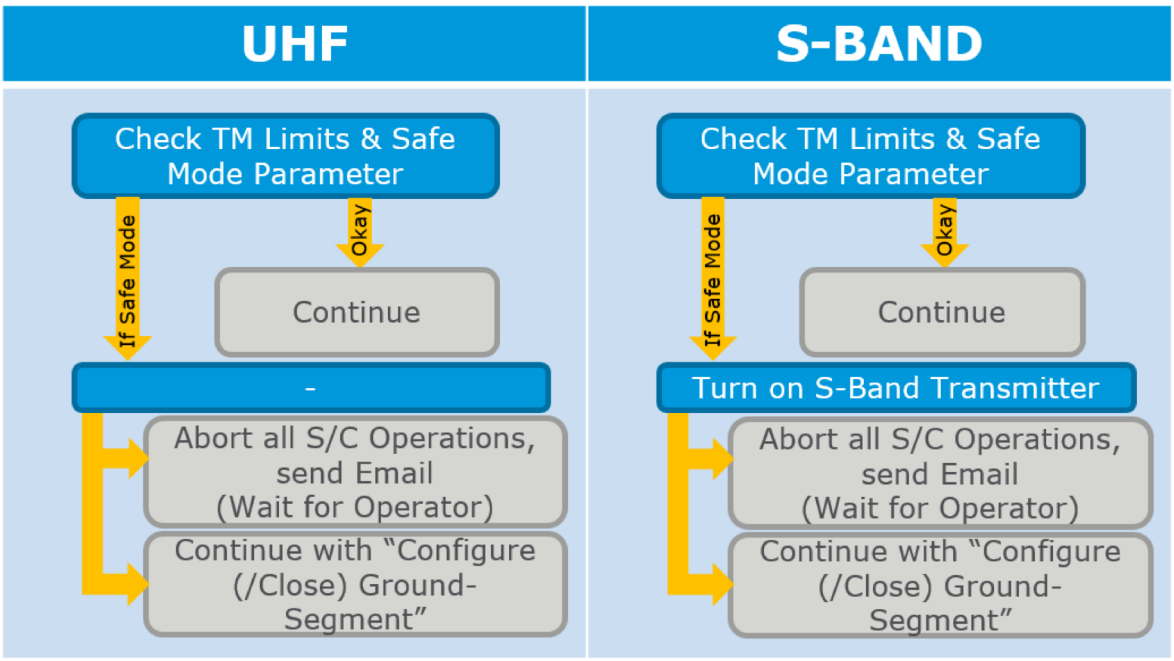


Figure 5.7: Detailed pass events for phase "Check Limits".

(iv) Active Interaction with S/C

This phase is crucial for interacting with the spacecraft. All effort and preparation of planned events during a pass is carried out here. Therefore, even though phases one, two, five and six depend on the frequency band used, everything but phase three and four can be seen as overhead, necessary to ensure safety of the satellite and execute the active interaction with the spacecraft. The chain of operations is depicted in Figure 5.8 below.

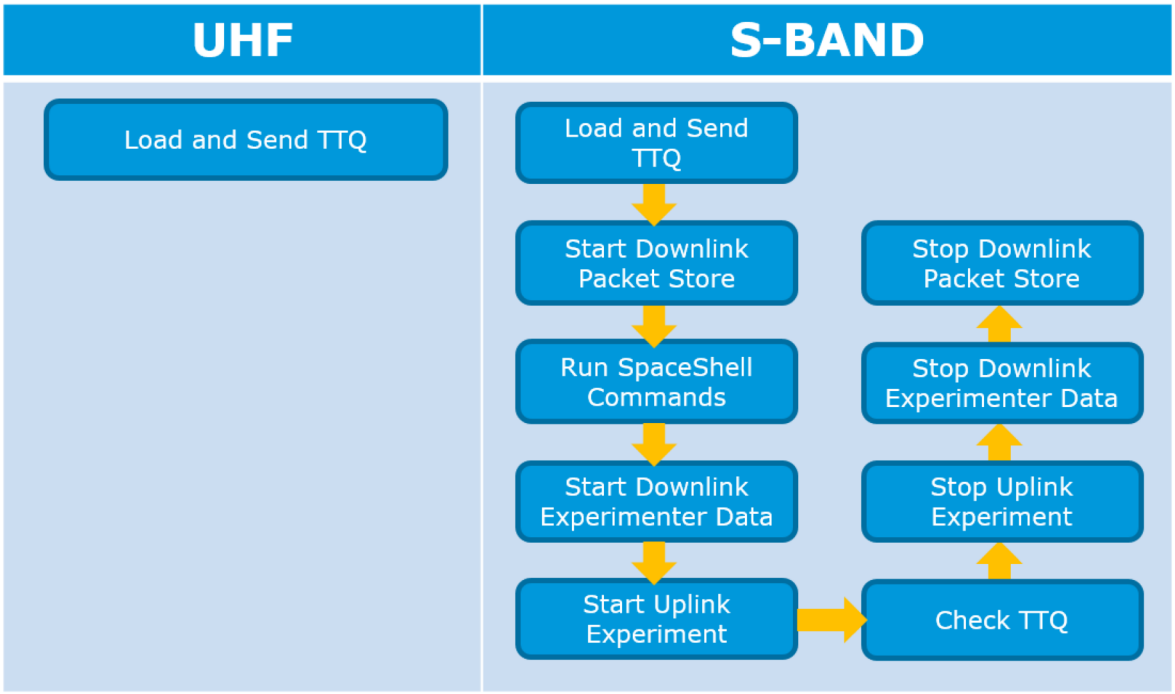


Figure 5.8: Detailed pass events for phase "Active Interaction with S/C".

(v) Configure (Close) Space-Segment

Before the link to the satellite is cut off due to its movement out of reach of the current ground station, the spacecraft is commanded to stop live telemetry and to start saving it in the Packet Store. Additionally, the current transmitter is turned off to conserve power and to keep the transmitter inside of the thermal range, increasing its lifetime. This process is shown in further detail in Figure 5.9.

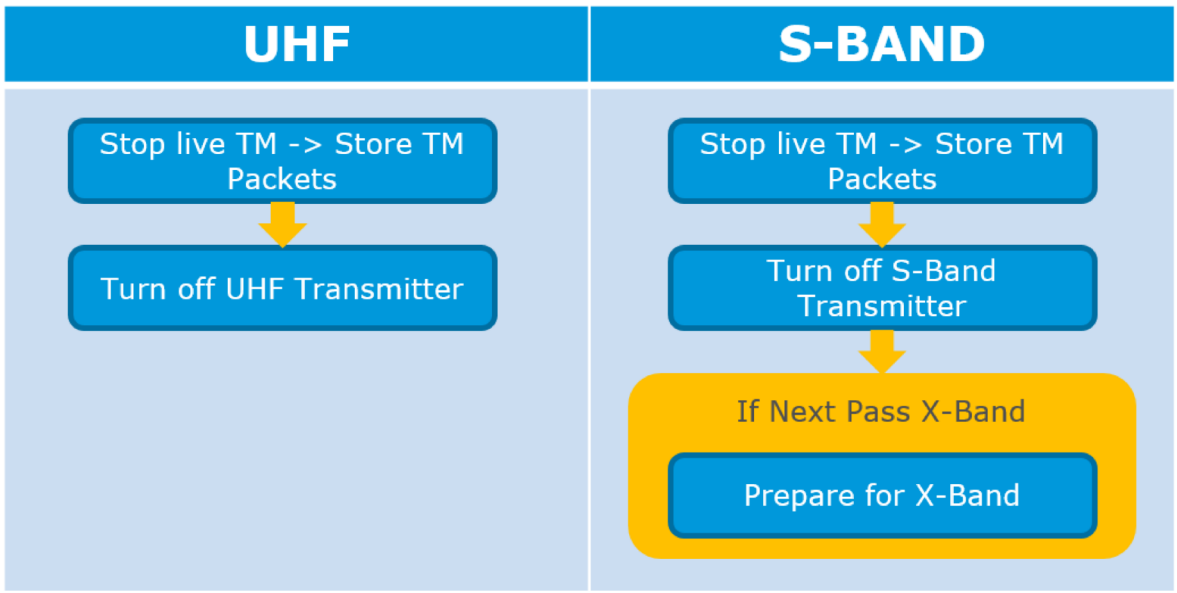


Figure 5.9: Detailed pass events for phase "Configure (Close) Space-Segment".

(vi) Configure (Close) Ground-Segment

At last, the connection to the NIS is closed and the ground segment is set to stand-by, concluding this pass. This final process is shown in Figure 5.10.

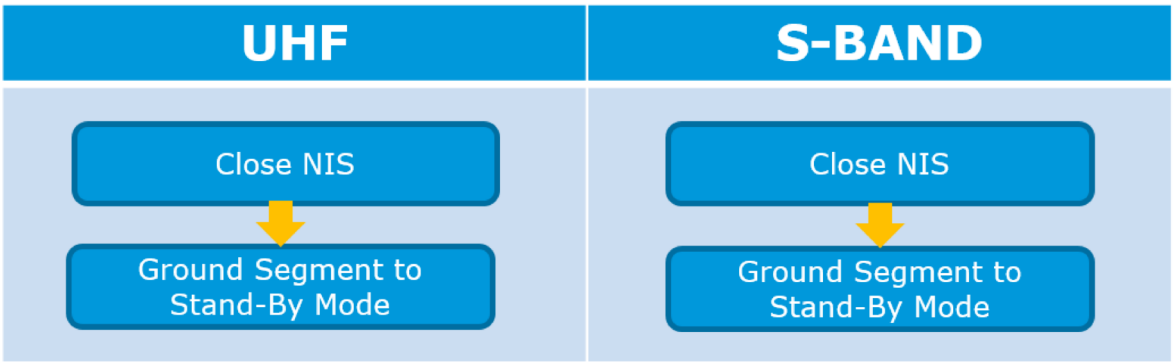


Figure 5.10: Detailed pass events for phase "Configure (Close) Ground-Segment".

5.3 Information Flow

In order to run the complete automation machinery with all its peripherals, information flow between programmes is a key element of the system. An overview of the most important players and their connectivity in the automation system can be seen in Figure 5.1 in the beginning of the chapter. To guarantee a running system, information flow shall occur without discontinuity. In OPS-SAT, many systems work together to enable automation. In this section, the exact data flow to and from the most important peripherals is presented and explained.

The flow of information will start at the NORAD TLE data, continue into the Mission Planning, followed by MATIS, SMF, SCOS, its sub-programmes and finally reach the spacecraft.

Table 5.2: Input-Output flow between programmes in automation chain.

System	Input	Output
Pass Planning	TLE<-NORAD (orbit information)	Schedule File with AOS/LOS time
Mission Planning	Schedule File with AOS/LOS time	- Stack of commands (into SCOS) - MAES, MAUS (saved in IN-TRAY Repository in MATIS) - Experiment files for FMS
MATIS	MAES, MAUS	Commands to: - SCOS (through SMF) - SMF - SpaceShell - Scripts - External Programmes
SMF	Commands from MATIS	- Commands to SCOS - Emails
REALS	Events and Out of Limits from SCOS	SMS to operation engineers
SCOS	Commands from MATIS through SCOS	- Log Files - Commands to Applications
FMS	Files for upload/download	- S/C: Files - Ground: Experimenter Files
Manual Stack	Commands from SCOS	Commands to Spacecraft

The first software involved is the Pass Planning, getting its orbital TLE from NORAD via the internet. In the NORAD system, the TLEs are saved as a text file from which the Pass Planning automatically extracts the corresponding TLE for OPS-SAT. It calculates the AOS and LOS times and saves this information in another text file. The Mission Planning System uses this file as an input and calculates the possible actions for each satellite pass. Depending on the power constraints, eclipses, number of experiments and resources on board of OPS-SAT, the MPS can create a Stack of commands, MAES and MAUS and can prepare the experiment files for transfer via FMS.
It is highly important to note the following. The generation of a Stack of commands should be avoided whenever possible, since sending a Stack of commands will not give feedback over the actual execution status of each single command itself, but gives only feedback whether the Stack was received by the spacecraft of not. This makes mission operation more difficult, since the execution status of these commands will not be visible by default for MATIS.

The MAES and MAUS are saved inside of the IN-TRAY folder in MATIS, where they are loaded automatically. As mentioned in Section 2.1.1, MAES has to be loaded before the MAUS to synchronise the tasks with the correct events. In MATIS, a lot of outputs are created and sent to different programmes. It contains commands to SCOS,

SMF, SpaceShell, scripts and external programmes. Commands to SpaceShell, scripts and external programmes are either triggered by MATIS writing a bash script on a certain Linux machine and its execution through MATIS in a later process, or directly by the the command line. Additionally, with the help of user defined Groovy functions in MATIS, other programmes can be triggered and the full range of the programming language Groovy can be exploited, extending the functionality infinitely. Nevertheless, in OPS-SAT, this is reduced to a minimum and functionality is added by creating scripts for simplicity reasons.
In the SMF, different drivers are triggered to execute the commands coming from MATIS. Thus SMF commands and controls SCOS and its peripherals, sending commands to the spacecraft. It triggers functionality and forwards subscribed telemetry, events and history back to MATIS. Additionally, SMF can send emails, write, create and execute files, execute command line actions and handle general information flow between programmes via its dedicated drivers. More detail is available in Section 2.1.3.
Commands coming from SMF to SCOS are responsible for the correct distribution of information flow to the different applications in SCOS. One of the most important applications is the Manual Stack commanding and configuring the spacecraft, ensuring its safety and correct functionality.
During the runtime of SCOS, SCOS and other systems constantly check for Events and Out of Limits (OoL) of telemetry data. Whenever a certain event is triggered or a telemetry parameter is OoL (for soft or hard limits see Section 2.1.11), the REALS system itself gets triggered and sends the corresponding values and events as an XML-file to the REALS server. The server then sends SMSs to the responsible persons and groups, notifying them about the non nominal state. These contingency states are then resolved manually by an operator.
Parallel to the commanding in SCOS, two other applications can interact with the spacecraft. The first one is SpaceShell, interactively executing Linux commands on the spacecraft via a shell like interface. The second one is FMS which transfers files from ground to the spacecraft and the other way around. These applications can be run parallel. Nevertheless, in the beginning, only one of the applications will be able to run since a non coordinated usage could create problems on the bus system of the spacecraft. Later, these operations can be run simultaneously, enabling more interactions during a pass.
In SCOS and MATIS, log files are created that can be filtered by a script to search for interesting logs and events, an important step to debug and track the automation process.

Chapter 6

Testing

This chapter is concerned with testing of the operation automation system. As mentioned beforehand, testing is a crucial part in space operations. An only briefly tested system can lead to unforeseen errors, endangering the mission.

SVTs - System Validation Tests

During OPS-SAT development, different System Validation Tests (SVT) take place before the spacecraft is able to be launched.
The contents of the different SVTs are the following:

- SVT-0: TM/TC-Chain, nominal procedures, SCOS operation, MPS, MIB [32]
- SVT-1: FDIR and safety [33]
- SVT-2: Experiments, payloads and automation [34]
- SVT-3: Safety, switching between flight computers, experimental payloads and TM parameters

These SVTs are necessary and executed and evaluated by the OPS-SAT team itself. The reason for this is the availability of the whole OPS-SAT team, including the University of Graz, who is responsible for the satellite integration and therefore has the Flight Model (FM) and the Engineering Model (EM). The direct and supported connection to the FM and EM, which is established during SVTs, allows quick testing of procedures, hardware components, software integration, new software updates and functionality, correct and efficient ground segment usage, testing of operation automation and correct transmission of TM and TC packets.
The SVTs simulate real mission operation conditions, like when the spacecraft is orbiting the earth, to the greatest possible extent. Therefore, in comparison to normal testing during development outside of the SVTs, the tests do not only consist of one specific

peripheral of the system, but moreover covers a huge variety of components and system elements working together. This utilisation of many components at once is important to find potential bugs hidden inside their interfaces and application interactions.
This thesis focuses on the automation and the peripherals connected to it. Therefore, a list of bugs, issues and their solution can be seen in Table 6.1.

Individual Testing

Despite the SVTs, the system is tested on a daily basis. Especially for PLUTO procedure concepts, the in-house Flatsat is sufficient. Therefore, around 90% of the actual automation system can be checked without the FM or EM. During daily tests, the following flaws and bugs have been identified and are listed inside of this section.

Table 6.1: Bug and issue report matrix with resolution and status (1/2).

#	**Description**	
	Resolution	**Status**
1	MATIS Server does not compile when parameters start with digit	
	Temporary: "~" written in front of parameter Final: New MATIS patch (JIRA)	Temporary
2	MATIS editor complains when "~" in front of parameter	
	Final: MATIS Editor update	Solved
3	MATIS editor complains when comparing two Booleans	
	Final: MATIS Editor update	Not solved
4	MATIS Server does not allow commands with repeating arguments (e.g. M040603b)	
	Temporary: Use LoadAndAuthoriseCommand function from CommandInjection_BD driver Final: New MATIS Server patch	Solved
5	Using Boolean (FALSE/TRUE) as input in command in MATIS is not correctly converted to Boolean in SCOS (0/1)	
	Temporary: Create calibration curve in MIB and use engineering value for parameter Final: MATIS Server patch	Temporary
6	EGSE gives random data when script "START_DOWNLINK.sh" is triggered twice	
	Temporary: Check in task manager if running, before execution Final: Fix in EGSE software	Temporary
7	Stacks cannot be loaded and sent from MATIS	
	Installation of corresponding drivers	Solved
8	Not able to start NIS application from task manager in MATIS	
	Installation of drivers, Modification of SCTLconfig.xml to add the be able to start and stop the application	Solved
9	After a certain time MATIS freezes during its file checks	
	Activate clean-up of MATIS history and temp files ->In MatisServerConfig.sta change matis.db.matisLocalDatabase.history.retention.limit to value between 10 and 200	Not confirmed
10	Data Proxy mode change not successful when switching to same mode	
	Patch for Data Proxy	Solved
11	No mode in Data Proxy for standby available	
	Implementation of Idle mode into Data Proxy	Solved
12	MATIS cannot execute SMF actions after some time	
	Maximum number of open SMF sessions has been reached. Therefore, number was increased from 5 to 200 by editing max_session_open in SMFconfig	Solved
13	SMON server status and button not visible in SCOS application launcher	
	Modification of TKMA and MMIconfig file to add functionality and user interface visibility	Solved

Table 6.2: Bug and issue report matrix with resolution and status (2/2).

#	Description	
	Resolution	**Status**
14	Sending email over SwissKnife tool not possible in MATIS	
	Port was opened to another machine (called OSREDMINE) to enable email communication through the internet by using the correct ports defined inside of the SwissKnife driver	Solved
15	Functionality to send SMS not available with the current setup	
	System integration of REALS as described in Section 2.1.11 responsible to send SMS without MATIS in the loop is implemented.	Ongoing
16	MATIS calendar driver not working	
	New drivers installed and configured in MATIS	Solved
17	Dynamic PTV overwrite (commanding without TM flow) not working	
	MATIS update (JIRA: MATIS-242)	Ongoing
18	Wrong time displayed in Calendar of Operation Manager	
	Temporary: Set default timezone on Linux machine to UTC and default date too. Final: MATIS Operation Manager patch	Temporary
19	Procedure with procedure dependency is executed before procedure it is dependent on	
	Temporary: Set always an earliest starting time or event in addition to procedure dependency. Final: MATIS Server patch	Temporary
20	Only validated procedures are transferred to operational mode. But validation not possible for procedures when "errors" are displayed in Editor	
	Temporary: Change configuration of MATIS Designer to export also unvalidated procedures. Final: Update of Editor to not mark code incorrectly as wrong.	Temporary
21	TM subscription from SCOS to MATIS not working	
	1) Implementation of missing drivers 2) TM subscription server (SMON) has to be set up 3) Buffer overflow of SMON when subscribing to many parameters	1), 2) Solved 3) Ongoing
22	SMON server subscription very slow	
	Change from "dedicated" to "shared SMON server". Therefore, server always on and does not need to shut down and start again. Note: Sometimes still slow, might be related SMON buffer overflow	Ongoing
23	MATIS Server does not start when certain procedures are compiled during start-up	
	Hints to major problem in MATIS Server software. First occurrence ever. Experienced in OPS-SAT repository during excessive testing of the system. It appears to be an endless JAVA loop inside of MATIS Server's software.	Investigation

These results are outcomes of this thesis work and are important for understanding the weaknesses and limits of the current automation system. Additionally, these findings were reported to the MATIS support and will therefore, be fixed in future releases. This is beneficial not only for OPS-SAT but for all other missions using MATIS as well. Especially, issue number 23 blocks mission operation automation completely and has to be solved by the MATIS software development team. The most important outcomes regarding the AS and its procedures are described in the following section. These outcomes shape the current and future procedures and approaches significantly due to new insight into the system.

The AS was not able to send commands to the spacecraft when no TM flow was coming from the spacecraft to the MCS. Nevertheless, this is needed to turn on the satellite's transmitter in the first place to receive data. Therefore, a function called "dynamic ptv override", which enables to send commands without TM flow, is now implemented for commands that do not expect TM flow.

During testing, it was realised that the automation system in combination with its connected peripherals is not yet perfectly stable. A system requirement is to send notifications to the operation engineers in case of an emergency, in case of OoL or in case of the spacecraft being in safe mode. Even though it is possible to send emails from MATIS to the operation engineers, a second external system, directly connecting to SCOS, is introduced and currently implemented into the system. This programme, capable of sending SMS to operation engineers regardless of the state of SMF or MATIS, is the REALS system and described in Section 2.1.11. Additionally, OPS-SAT's procedure flow has been influenced significantly. In the past, the concept was designed to execute procedure after procedure in a MAUS, triggered by the event of the MAES. Nevertheless, the current approach is a parallel procedure running during a pass, checking for OoL and reacting accordingly.

Furthermore, if needed, procedures shall check whether TM flow is available or not and react to it as well. In addition, due to the testing it was discovered that many drivers for SMF were wrongly configured or missing. Missing drivers lead to missing functionality, which would therefore lead to an unusable automation system. Detailed documentation of testing can be found in Appendix D. Due to this intense testing, operation automation can further be implemented and accelerated.

Chapter 7

Conclusion

The research and the experiences during this thesis confirmed the assumption of a clear lack of operation automation in the overall area of mission operations. Currently, only the passes of PROBA satellites operated from ESEC in Redu are fully automated, saving these missions a lot of resources during operations. These resources can be invested in software improvement and technology. At ESOC, GAIA obtains the largest amount of experience with operational automation during passes to this point. The Sentinel satellites and ExoMars currently begin to realise their automation with MATIS, too. Nevertheless, OPS-SAT will be the first mission to use fully autonomous operations during non-supervised passes.
In this section, the overall achievements of this work are presented. Additionally, this chapter compares the target of this thesis with the reached objectives, giving a critical overview. Furthermore, a glimpse of the future of OPS-SAT's operation automation development and a general view of the future of space mission operation automation is presented in the end of this chapter, completing the thesis.

7.1 Achievements

During this master thesis, several achievements have been accomplished, some even exceeding the original objectives. Many of them are mission specific, while others are findings applicable to automation of operation and testing in general.
Mission specific acquisitions are the three different converters, the new standard for Excel procedures, the setup of the automation system, the PLUTO procedure integration for operation and testing and the findings of bugs in MATIS as well as its reports and partial improvement. Even though these achievements were OPS-SAT specific, they can mostly be applied to other ESA missions. Especially the bug reporting for the mission automation system MATIS and its peripherals are bringing MATIS, and its reputation in the world of mission operation, a step towards an established, reliable and used tool for automation. Additionally, depending on the missions' needs, the Excel

procedure standard can be reused with the corresponding converters for other ESA missions. Missing functionality in the standard for upcoming missions can always be added on the fly when the need emerges.
General results of this work include the concept of mission automation, the knowledge regarding automation of procedures for new missions and the conclusion for future automation designs due to exemplary hardware testing with OPS-SAT.
Due to this, future missions can build their systems automation-compatible to all systems and software from the start, knowing where the challenges for the automation of operations lie within old ground systems, hardware and the general interaction and information flow between programmes.
The last issue in particular is very important and often overlooked during the first mission design, due to the misunderstanding of its priority. Nevertheless, a well structured and planned information flow and well designed interface between the programmes can save a lot of time, money and resources. The time needed to apply workarounds to make information flow possible should not be underestimated.
To conclude the achievements, this thesis covered the automation concept for OPS-SAT, the converters and new standards for Excel procedures, the setup of the automation system and surrounding peripherals, testing of the MATIS system and connected programmes, its software improvements for all missions and the start of the REALS system integration for OoL alerts via SMS to the operation engineers. Furthermore, during this thesis, support for SVT-2 and documentation for the Quality Assurance Review (QAR) has been provided and was accepted in reviews as official documentation.

7.2 Pros and Cons of the System

From the thesis work, the following issues concerning pros and cons of the analysed system arise.

MATIS and SMF

The decision on using MATIS as a mission automation system has delivered a high amount of functionality, benefits and improvements. Additionally, testing and executing of procedures is now possible at the push of a button instead of a long process of manual preparation and complicated user input with a high possibility for human errors. All of these inconveniences have been eliminated, giving the operator a high-level interface to command the spacecraft.
Nevertheless, with the introduction of MATIS and the utilisation of SMF, an additional amount of complexity is introduced to the system. Since MATIS has not been used extensively for unsupervised passes, this functionality has not yet been tested during operational space missions. Additionally, the automation system in combination with SMF is not absolutely stable when running over a long period of time and requires

a restart every now and then. Furthermore, the configuration of MATIS and SMF is not straight forward and needs external expertise from the mission support. This complication consumes a lot of time, especially because OPS-SAT is using an old version of Linux server (SLES 11) instead of the newer version (SLES 12). As a result of this, OPS-SAT has to wait for additional support to implement a new version of MATIS.
Furthermore, the configuration of drivers in MATIS (cf. Section 2.1.1) and SMF (cf. Section 2.1.3) has played an important role in this thesis. Nevertheless, it requires a deep understanding of the system environment and its overall structure in the system configuration on file level on two virtual machines (OSMATx and OSMCx) as well as the setup of MATIS in the MATIS Designer and its interface. Even though the system configuration and manuals have been written down for internal use at ESA-ESOC during this thesis, it has to be considered that external users do not have access to this document and will either need to investigate a lot or pay for MATIS support directly.

Converters

Even though the converters will reduce the amount of time needed to implement Excel procedures and their corresponding se.xml files significantly, some converters contain minor flaws which need further improvement.
Whereas no flaws of the "SCOS to MATIS MISC MIB Converter" have been detected yet, the "Excel Procedures to MATIS System Element Configuration File Converter" currently allows undefined types of input arguments inside of Excel procedures and defines unknown or undefined input argument types as "STRING" automatically. This procedure has been implemented to enable integration into MATIS even though a procedure input argument type is not defined or known, allowing faster development. Nevertheless, this is a risk for operational use and shall be disabled before operational execution of procedures with input arguments.
The "Excel Procedures to PLUTO Code Converter" is capable of converting the defined standard as described in Section 4.1. Undefined commands will only be implemented as a comment inside of the code. Additionally, certain functions like "MAL response" are not working yet due to missing functionality in MATIS. A further constraint is the function "GOTO", which is not implemented yet and has to be edited manually after the code has been generated.
Despite these limitations, the usage of these converters reduces the implementation time significantly. A detailed comparison and further information about the test procedure is presented in Section 4.1.

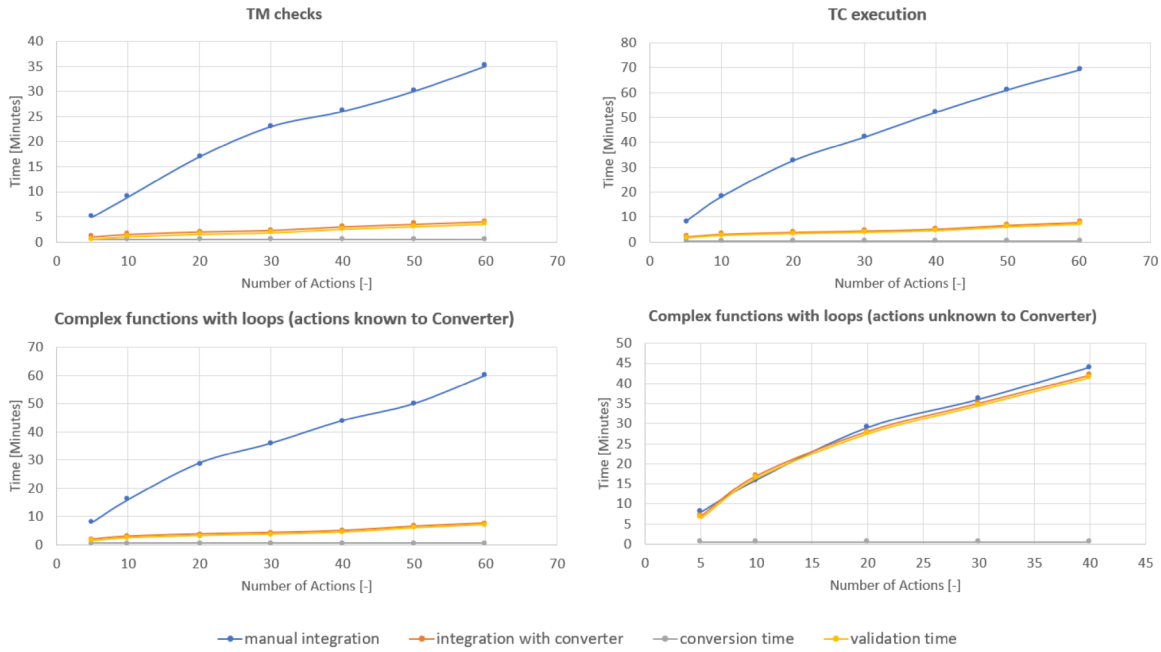


Figure 7.1: Duration comparison between manual implementation and automatic conversion with review and correction.

Scaling these results to all procedures provides an approximate time saving of 80% in comparison to manual insertion. The data was procured with only a small amount of data samples, nevertheless, a clear trend is perceptible. Additionally, the curve connecting the data points is an polynomial interpolation between the data points and was implemented for better visualisation, but does only represent an approximation.

7.3 Lessons Learned

In project development, unforeseen complications and sub-optimal solutions occur while the project evolves: OPS-SAT is no exception. A number of lessons learned during OPS-SAT's way to orbit with regard to operation automation are listed below.
Starting with the software and operating system used, it reveals that an upgrade of the operating system from SLES 11 to SLES 12 would have been a good investment at the beginning of the project. The usage of SLES 11 comes with the downside of MATIS needing a specific patch to be executable on SLES 11, since the default MATIS development environment is based on SLES 12. This adds effort when debugging is necessary or a new patch for MATIS has been developed.
Additionally, a new version of SCOS is available which interacts with MATIS in a more stable way due to improvements on the connectivity to SMF. Furthermore, this new version adds further functionality to the system. Therefore, with an update to SLES 12, an update to the newer version of SCOS should be implemented. Nevertheless, this will take time since the system and all its programmes will have to be tested again.

Initially, one or two weeks dedicated to the setup of MATIS by an expert would have been extremely useful instead of trying to set it up oneself and would have saved a lot of time during development of the automation system.
During the mission automation pass planning, it was perceived that X-Band cannot be triggered by the NanoMind, the on-board computer which gets input signals from ground. This results in a logistic problem: The X-Band cannot be triggered by the operator directly from ground, but requires an additional experiment running on the SEPP which triggers the X-Band downlink. This limitation complicates X-Band passes significantly and shall be avoided in future missions.
Taking all this into consideration, a generalisation for other missions adapting to automation can be derived. A study of all programmes and systems with their requirements and future upgradability shall be executed in the beginning of the project and again at certain stages of the project when more information is available. This is an additional effort, especially for the system administrator setting up the required software. Nevertheless, it saves an enormous amount of time during development whenever workarounds have to be applied to get new configurations running on the old system. In OPS-SAT, this upgrade will be performed when the satellite is in orbit, when operations are running stable and the team is confident in updating the ground system.

7.4 Conclusion

In summary, this work provides the basics for mission automation. On the setup side it contains the configuration of the automation software MATIS and SMF with its drivers. Regarding automation, this work reflects the basic concept of autonomous passes and the corresponding high-level procedures for ground segment setup. Additionally, due to this thesis and its converters, implementation of the Excel procedures written by the operation engineer into PLUTO procedures is more efficient. As a result, it is saving ten thousands of lines of code to be written and hundreds of hours of work for the engineer to implement these procedures. In addition, it explains the sequence of transferring the MIB to MATIS and provides the first manual for the usage of MATIS and its systems, enabling future projects to catch up quickly with the progress OPS-SAT delivered during its approach to full pass-operation automation.
Furthermore, during this thesis, a standard for Excel procedures has been developed to match the expectations of the programmed converters resulting from this thesis. Therefore, other missions can adopt this standard and further add new functions if needed, providing a unique tool which enables readability for humans due to the Excel procedures and quick implementation for automation.
As a result, the full extent of this work will not only result in faster development and automation implementation for the OPS-SAT mission, but is also meant to accelerate

the adaption from manual operations to autonomous operations eventually, leading to faster and more efficient ground operations and a significant reduction of operational costs.

7.5 Glimpse into the Future

During the development of this thesis, a clear movement towards automated operations has begun. New and established missions are directing resources towards mission operation automation systems. MATIS in particular gained a raise in reputation and popularity along the ranks of operation engineers.
Additionally, the retirement of several experienced operation engineers in the near future implies the necessity of creating new systems to fill the arising lack of manual expertise. With NewSpace companies forming and their competition for affordable systems among each other, automation of operation is currently a hot topic in the field of satellite operations.
Furthermore, ESA and NewSpace companies desire to reduce the costs for satellite passes significantly while increasing the functionality of the systems at the same time. Therefore, another movement has started: The transition from static legacy systems to scalable cloud based solutions. Cloud based ground stations allow ground station software and processing capacity to be deployed when needed. Combining these technologies with an SDR solution provides great configurability, a minimum amount of hardware to maintain, and a significant reduction of operational costs. Furthermore, when working with external customers, they can plan their passes via web interfaces and the data can be processed and sent to the customers automatically without operation engineers and data analysts in the loop. Ultimately, with the rise of machine learning in space industry, these tools can be utilised directly in the cloud and data can be generated more efficiently, saving costs and resources. Pioneers in this new field are Telespazio Vega with their ENABLE system[1], Amazon Web Services (AWS)[2], Vision Space[3] for cloud based mission control systems and Arctic Space Technologies (AST)[4] for cloud based ground station operation.
To conclude, the space industry is currently evolving at a rapid pace and due to automation and flexible cloud based technologies, continuous mission operation will become significantly more favourable.

[1]ENABLE website: https://www.telespazio-vega.de/en/solutions-services/enable
[2]AWS website: https://aws.amazon.com/
[3]Vision Space website: https://www.visionspace.com/
[4]Actic Space Technologies website: http://www.arcticspacetech.com/

Bibliography

[1] A. A. Siddiqi and R. Launius, "Deep space chronicle: A chronology of deep space and planetary probes 1958-2000," Jul 2002. [Online]. Available: https://history.nasa.gov/monograph24.pdf

[2] J. Kauffman, "A successful failure: Nasa's crisis communications regarding apollo 13," *Public Relations Review*, vol. 27, pp. 437–448, Dec 2001.

[3] J. Foust, "Spacex's space-internet woes: Despite technical glitches, the company plans to launch the first of nearly 12,000 satellites in 2019," *IEEE Spectrum*, vol. 56, no. 1, pp. 50–51, Jan 2019.

[4] I. Levchenko, M. Keidar, J. Cantrell, Y.-L. Wu, H. Kuninaka, K. Bazaka, and S. Xu. (2018, Oct) Explore space using swarms of tiny satellites. [Online]. Available: https://www.nature.com/articles/d41586-018-06957-2/

[5] F. Teston, R. Creasey, and J. Van der Ha, "Proba: Esa's autonomy and technology demonstration mission," 48th International Astronautical Congress, Oct 1997.

[6] K. Gantois, F. Teston, O. Montenbruck, P. Vuilleumier, and P. van Braembusche, "Proba-2 mission and new technologies overview," Small Satellite Systems and Services - The 4S Symposium, Dec 2006.

[7] J. Llorente, A. Agenjo, C. Carrascosa, C. de Negueruela, A. Mestreau-Garreau, A. Cropp, and A. Santovincenzo, "Proba-3: Precise formation flying demonstration mission," *Acta Astronautica*, vol. 82, no. 1, pp. 38 – 46, 2013, 6th International Workshop on Satellite Constellation and Formation Flying. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576512002202

[8] M. Francois, S. Santandrea, K. Mellab, D. Vrancken, and J. Versluys, "The proba-v mission: the space segment," *International Journal of Remote Sensing*, vol. 35, no. 7, pp. 2548–2564, 2014. [Online]. Available: https://doi.org/10.1080/01431161.2014.883098

[9] G. Paolo Calzolari, Y. Doat, S. Haag, C. Haddow, M. Pecchioli, and E. Sorensen, "Automation of esoc mission operations," in *Automation of ESOC Mission Operations*, Jun 2006.

[10] D. Milligan, A. Rudolph, G. Whitehead, and T. Loureiro, *Gaia Ground Segment Development and Operations Concept.* [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2010-2162

[11] P. Kretschmar, M. Küppers, R. Walker, D. Evans, and OPS-SAT team, *OPS-SAT summary Launchtime Presentation 2019.*

[12] ESA, *ECSS-E-ST-70-32C – Test and operations procedure language*, ESA Std., Jul 2008. [Online]. Available: https://ecss.nl/standard/ecss-e-st-70-32c-test-and-operations-procedure-language/

[13] MATIS Development Team (European Space Agency), "Matis overview user manual," Jan 2019, eSA internal Reference ID: EGOS-MCS-MAS-SUM-0001.

[14] A. Loretucci, D. D. Nisio, G. Ugolini, M. Kirchmann, and A. Simonic, "Mission automation system - system level add," Nov 2012, eSA internal Reference ID: EGOS-MCS-MAS-ADD-0001.

[15] MATIS Development Team (European Space Agency), "Matis designer user manual," Jan 2019, eSA internal Reference ID: EGOS-MCS-MAS-SUM-0002.

[16] M. Kirchmann and A. Loretucci, "Mission automation system schedule interface control document," Nov 2012, eSA internal Reference ID: EGOS-MCS-MAS-ICD-0002.

[17] D. D. Nisio, "Mission automation system schedule event interface control document," May 2014, eSA internal Reference ID: EGOS-MCS-MAS-ICD-0004.

[18] ESA, *ECSS-E-ST-70-31C – Ground systems and operations - Monitoring and control data definition*, ESA Std., Jul 2008. [Online]. Available: https://ecss.nl/standard/ecss-e-st-70-31c-ground-systems-and-operations-monitoring-and-control-data-definition/

[19] D. D. Nisio, "Mission automation system smf interface control document," May 2014, eSA internal Reference ID: EGOS-MCS-MAS-ICD-0003.

[20] SCOS-2000 Development Team, "Getting started with scos-2000 development environment," May 2010, eSA internal Reference ID: EGOS-MCS-S2K-SUM-0003.

[21] *SPACE LINK EXTENSION—INTERNET PROTOCOL FOR TRANSFER SERVICES*, CCSDS - The Consultative Committee for Space Data Systems Std., Sep 2015. [Online]. Available: https://public.ccsds.org/Pubs/913x1b2.pdf

[22] *CCSDS FILE DELIVERY PROTOCOL (CFDP)*, The Consultative Committee for Space Data systems Std., Jan 2019. [Online]. Available: https://public.ccsds.org/Lists/CCSDS%207270P42/727x0p42.pdf

[23] S. Cooper, *CCSDS MO in Space*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2012-1291637

[24] Coelho, C., Evans, D. (ESA), and Koudelka, O (Technical University of Graz), "Ccsds mission operations services on ops-sat," Apr 2015. [Online]. Available: https://www.researchgate.net/publication/275639081_CCSDS_Mission_Operations_Services_on_OPS-SAT

[25] EGOS. (2019, Jun) Scos documentation. [Online]. Available: file://esaad/esoc/OPS-GI-CurrentRef/SCOS-2000/SCOS-2000%20R5.4.21/Documentation/S2K/

[26] C. Coelho, "A software framework for nanosatellites based on ccsds mission operations services with reference implementation for esa's ops-sat mission," Ph.D. dissertation, Nov 2017.

[27] A. Lofaldli and G. Smolinski, "Smile ground station icd," May 2019, eSA internal Reference ID: ESA-SMILE-GST-ICD-0001.

[28] Orbit-CS. (2019, Jul) Gaia100 antenna. [Online]. Available: http://orbit-cs.com/wp-content/uploads/sites/3/2017/08/Gaia100.png

[29] G. Scotti, "Network interface system - software user manual," Oct 2010, eSA internal Reference ID: EGOS-NIS-NIS-SUM-1001.

[30] "Jira - an issue and tracking software by atlassian," https://www.atlassian.com/software/jira, Accessed: 12th of July, 2019.

[31] SCOS-2000 Development Team, "Software user manual misc subsystem," Apr 2007, eSA internal Reference ID: EGOS-MCS-S2K-SUM-0005.

[32] D. Evans, "Ops-sat svt preliminary test report," ESA-ESOC, Test Report, Apr 2017, eSA internal Reference ID: ESA-DOPS-MGT-TR-0001.

[33] D. Evans, "Ops-sat system validation tests plan," ESA-ESOC, Test Report, Nov 2016.

[34] D. Evans, "Ops-sat svt-2 test report," ESA-ESOC, Test Report, May 2019.

[35] C. Haddow, C. Müller, G. Scotti, and T. Ulriksen, *Network Interface System (NIS); ESAs Next Generation CCSDS SLE Interface System.* AIAA, 2006, p. 7. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2006-5732

[36] D. D. Nisio, "Matis documentation control list," Jan 2019, eSA internal Reference ID: EGOS-MCS-MAS-CIDL- 0001.

[37] D. D. Nisio and G. Ugolini, "Matis cg matis configuration and installation guide," May 2014, eSA internal Reference ID: EGOS-MCS-MAS-CIG-0001.

[38] M. Kirchmann and A. Loretucci, "Mission automation system procedure icd interface control document," Nov 2012, eSA internal Reference ID: EGOS-MCS-MAS-ICD-0001.

[39] A. Loretucci, "Mission automation system internal interface control document," May 2014, eSA internal Reference ID: EGOS-MCS-MAS-ICD-0005.

[40] D. D. Nisio and A. Loretucci, "Mission automation system procedure language icd," May 2014, eSA internal Reference ID: EGOS-MCS-MAS-ICD-0006.

[41] D. D. Nisio, "Mission automation system multi-context icd," Nov 2012, eSA internal Reference ID: EGOS-MCS-MAS-ICD-0007.

[42] MATIS Development Team (European Space Agency), "Matis operation manager user manual," Jan 2019, eSA internal Reference ID: EGOS-MCS-MAS-SUM-0003.

[43] MATIS Development Team (European Space Agency), "Matis server user manual," Jan 2019, eSA internal Reference ID: EGOS-MCS-MAS-SUM-0004.

[44] ESA and TERMA, "Matis training for operators," Jan 2019, training provided by ESA in cooperation with TERMA.

Appendices

Appendix

A Automation Concept Document - QAR

This document was submitted to the Quality Assurance Review of OPS-SAT and has been accepted as official documentation.
It contains the Automation Concept as stated in this thesis, as well as a list of frequently needed actions in MATIS, SMF and SCOS. Additionally, it provides information about the exact procedure to change the configuration and setup of the mentioned programmes. Furthermore, it describes how to implement new features into the SMF and MATIS.

esoc

European Space Operations Centre
Robert-Bosch-Strasse 5
D-64293 Darmstadt
Germany
T +49 (0)6151 900
F +49 (0)6151 90495
www.esa.int

OPS-SAT - Automation Concept

Prepared by **Felix Hessinger**
ESA
Reference **ESA-OSAT-MCS-DOC-0001**
Issue/Revision **1.1**
Date of Issue **14/03/2019**
Status **Draft**

APPROVAL

Title OPS-SAT - Automation Concept	
Issue Number 1	**Revision Number** 1
Author Felix Hessinger	**Date** 14/03/2019
Approved By	**Date of Approval**

CHANGE LOG

Reason for change	Issue Nr.	Revision Number	Date
Initial Version	1	0	14/03/2019
Update for Pass Prediction	1	1	05/06/2019

CHANGE RECORD

Issue Number 1	Revision Number 1		
Reason for change	**Date**	**Pages**	**Paragraph(s)**
Additional Information for Pass Prediction Section	05/06/2019	6	2.1

DISTRIBUTION

Name/Organisational Unit

Table of contents:

1 INTRODUCTION

The goal of OPS-SAT's automation is the significant reduction in operational costs and the automated operation during nominal passes outside of the official working hours. Additionally, the experiences gained from OPS-SAT automation is a step towards a standardisation of tested automation tools and software, future missions can build on.

In chapter 2 this document gives a rough overview about the systems and automation concept used for the OPS-SAT mission.
Chapter 3 describes the elements of the automation system following up with a quick-start manual in chapter 4 and ended with a collection of frequently asked questions regarding automation and the usage of the corresponding software.

1.1 Scope of the Document

The scope of this document is to define the automation concept for general mission operations, illustrated by the example of the OPS-SAT System. The concept covers the automation software and its necessary inputs, outputs and peripherals.

This document describes the conventions and design of OPS-SAT operation automation.
Additionally, this document shall give a first overview and quick-start for automation to be used by other missions.

1.2 Applicable and Reference Documents

1.2.1 Applicable Documents (ADs)

ID	Reference Name	Reference ID
[MAT-ICD-0001]	Matis - Mission Automation System Procedure ICD	EGOS-MCS-MAS-ICD-0001
[MAT-ICD-0002]	Matis - Mission Automation System Schedule ICD	EGOS-MCS-MAS-ICD-0002
[MAT-ICD-0003]	Matis - Mission Automation System SMF ICD	EGOS-MCS-MAS-ICD-0003
[MAT-ICD-0004]	Matis - Mission Automation System Schedule Event ICD	EGOS-MCS-MAS-ICD-0004
[MAT-ICD-0006]	Matis - Mission Automation System Procedure Language ICD	EGOS-MCS-MAS-ICD-0006
[ECSS-E-70-32]	PLUTO - Procedure Definition Language	ECSS-E-ST-70-32C
[MAT-SUM-0001]	Software User Manual - Overview	EGOS-MCS-MAS-SUM-0001
[MAT-SUM-0002]	Software User Manual - Designer	EGOS-MCS-MAS-SUM-0002
[MAT-SUM-0003]	Software User Manual - OpManager	EGOS-MCS-MAS-SUM-0003
[S2K-SUM-0003]	SUM – Getting Started	EGOS-MCS-S2K-SUM-0003
[S2K-SUM-2110]	SUM – General Desktop	EGOS-MCS-S2K-SUM-2110

[S2K-SUM-2232]	SUM – TM Packet History	EGOS-MCS-S2K-SUM-2232
[S2K-SUM-2330]	SUM – TC SPACON	EGOS-MCS-S2K-SUM-2330
[S2K-SUM-2310]	SUM – Manual Stack	EGOS-MCS-S2K-SUM-2310
[S2K-SUM-2360]	SUM – TC History Display	EGOS-MCS-S2K-SUM-2360
[S2K-ICD-0041]	ICD – TC SPACON	EGOS-MCS-S2K-ICD-0041
[S2K-ICD-0042]	ICD – TC History	EGOS-MCS-S2K-ICD-0042
[CMS-ICD-0001]	ICD – Command Supervisor	S2K-ADVMON-ICD-CS-0001
[MAT-TRN-0001]	MATIS – Training User	
[SpS-SUM-0001]	SpaceShell Summary	

1.2.2 Reference Documents (RDs)

ID	Reference Name	Reference ID
[MAT-ADD-0001]	Matis - System Level ADD	EGOS-MCS-MAS-ADD-0001
[MAT-ICD-0005]	Matis - Mission Automation System Internal ICD	EGOS-MCS-MAS-ICD-0005
[MAT-ICD-0007]	Matis - Mission Automation Multi-Context ICD	EGOS-MCS-MAS-ICD-0007
[MAT-SUM-0004]	Software User Manual – MATIS Server	EGOS-MCS-MAS-SUM-0004
[S2K-SUM-0004]	SUM – Time & Dates	EGOS-MCS-S2K-SUM-0004
[S2K-SUM-CS]	SUM – Command Supervisor	S2K-ADVMON-SUM-CS-0001
[S2K-SUM-2120]	SUM – Event Log Display	EGOS-MCS-S2K-SUM-2120
[S2K-ICD-0001]	ICD – Database Import	EGOS-MCS-S2K-ICD-0001
[S2K-ICD-0002]	ICD – Stack Import	EGOS-MCS-S2K-ICD-0002
[S2K-ICD-0004]	ICD - Telemetry Packet	EGOS-MCS-S2K-ICD-0004
[S2K-ICD-0005]	ICD - Command Injection	EGOS-MCS-S2K-ICD-0005
[S2K-ICD-0028]	ICD - Telemetry Replayer	EGOS-MCS-S2K-ICD-0028

2 OVERVIEW

The following figure shows the automation system's dependencies.

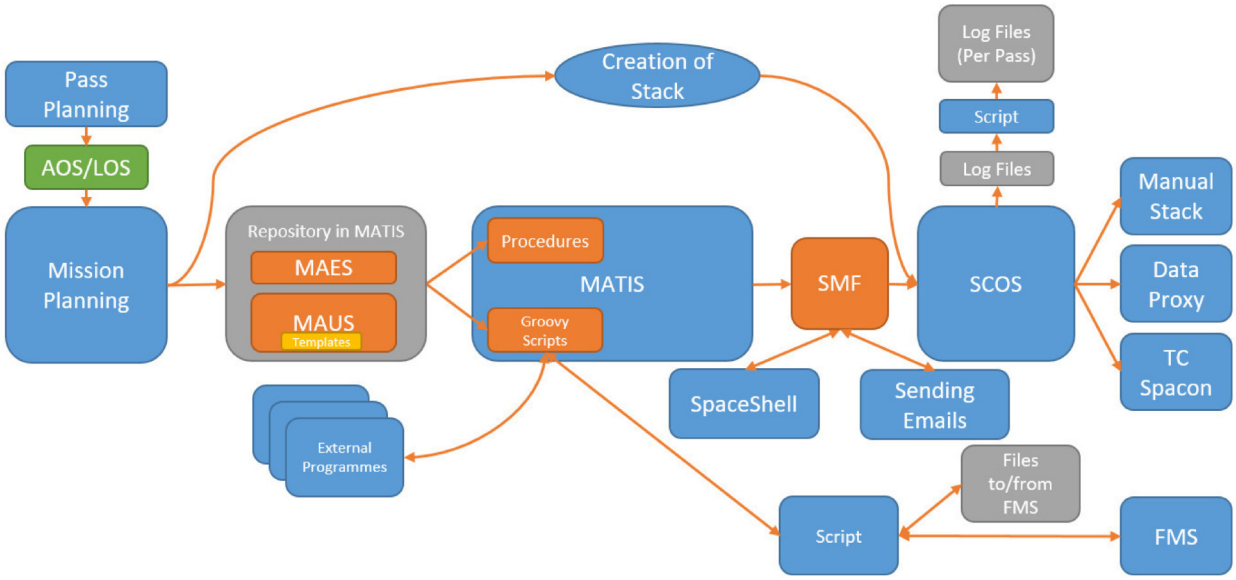


Figure 1: Dependency and flow chart of automation system

2.1 Pass Prediction

The input is a two line element (TLE) taken from NORAD which describes the orbit. The TLEs are automatically downloaded from https://celestrak.com/NORAD/elements/ inside of the python programme.
It uses an open source library such as PyEphem to compute the position and velocity of the spacecraft.
From a location of the used antenna on ground (ESOC or TUG) the program generates a list of events indicating start and end of a pass. These events are used to generate an XML file with the format MATIS expects.
The programme is planned to run on machine ESOC-1 STC and will automatically transfer the files via "scp" or "sftp" to the corresponding machine where the data for AOS and LOS is needed.

2.2 MATIS (Mission Automation System)

The automation of processes are coordinated by the Mission Automation System (MATIS). MATIS can execute MAUS, MAES, procedures and Groovy scripts. Procedures are written in PLUTO language (see [ECSS-E-70-32] for documentation) to enable readability for the operator. These are used for simpler tasks like sending telecommands, checking telemetry and reacting to its values or setting options in SCOS. The coordination with other programs using the PLUTO language in MATIS is done via drivers (SMF). These drivers can be added if more usability is needed.

Groovy scripts are written in Groovy language and have full capability of its programming language. Groovy scripts are used for complex tasks, interacting with external programs without a SMF driver or to add general functionality to MATIS.
A small introduction can be found in [MAT-SUM-0001].
To coordinate the execution of procedures MAES and MAUS are used. MAES are event schedules containing the event date, time, ID, name and other information about events. On execution of a MAES the containing events are loaded into the MATIS Calendar.
MAUS are user schedules which contain tasks containing linked procedures/activities. Each MAUS task can have dependencies of a prior execution of a task, a precondition which can be an event or a specific time. When a task is triggered by an event or time then after loading the MAUS these tasks will be aligned with these and executed when the preconditions and event or time is fulfilled.
This allows flexibility, repetition and reusability of a certain task when a MAUS is loaded. Therefore, a MAUS for e.g. a data dump of the satellite data could also be static and just called when needed.
MAUS and MAES will be created in the Mission Planning either by the operator or by a Mission Planning Software.

2.2.1 MATIS Server

The MATIS Server is the core component for the execution of the MATIS system. It runs as a process without user interfaces. Interaction and monitoring is possible through the Operation Manager. It will connect to the Server via a TPS/IP network.
Additionally, MATIS Server provides basic MATIS functionality and the interface to the SMF services. More information can be found in [MAT-SUM-0004].

2.2.2 MATIS Designer

The MATIS Designer is one of the components of MATIS. It is designed to be the main tool for editing MATIS projects. It provides full editing functionality and contains certain tools that are not available in the MATIS Operational Manager. More information in [MAT-SUM-0002].

2.2.3 MATIS Operational Manager

The Operational Manager is the main graphical user interface of MATIS Server. It gives an overview about the components running on the MATIS Server in the background.
Additionally, it contains the functionality of debugging and step by step execution of the PLUTO procedures. More information in [MAT-SUM-0003].

2.2.4 SMF

The MATIS Service Management Framework (SMF) enables the MATIS user to interact with other programs and with MATIS itself through SMF drivers translating between the programs. More information can be found in [MAT-ICD-0003]

2.3 SpaceShell

Space Shell is a C application allowing experimenters or the flight control team to communicate with the OPS-SAT Satellite Experimenter Platform (SEPP) like a Linux Shell and consists of a ground application (ground port) and a space application (space port).
Usage of the SpaceShell requires live communication with the satellite. In case of OPS-SAT this is only used in S-Band due to band width limitation in UHF and the missing X-Band uplink possibility.

2.4 Mission Planning

The Mission Planning is responsible for four tasks. These tasks are the creation of Stacks for SCOS, the MAES, the MAUS and shifting the correct files into a folder which's content will be uploaded to the spacecraft. These contents can be images for the SEPP or experiments.
The Mission Planning gets the AOS and LOS times in a XML file from the pass prediction/planning. With this pass data the Mission Planning will create additional events with their corresponding times depending on the executable tasks and resources available. These are stored in a new MAES. Additionally, it will fill a predefined User Schedule (MAUS) which will then be transferred to a specific folder in MATIS to enable automatic loading and execution of the MAUS.
The system is designed in a way that the Mission Planning can also be substituted by an operator.

2.5 SCOS

SCOS is the used mission control software. Further information about full functionality is given in the documents starting with ID S2K.
A first guide is given in [S2K-SUM-0003].
In the content of this document SCOS gets input from the MATIS over the SMF and handles the TC and TM management to and from the spacecraft, respectively.

2.5.1 Application Launcher

In the Application Launcher the different applications and services of SCOS can be controlled. The applications can be started, stopped and restarted by the colorful buttons in the bottom left corner. Additionally, the buttons "Log" and "Conn Status" can be used for logging and for the connection status of the application, respectively.
By going through the tabs on the top of this Application Launcher one can find the applications and services sorted by their category.

2.5.1.1 Manual Stack

In the Manual Stack commands and stacks can be loaded, armed and sent. Commands are used to control the spacecraft and to talk to it. Stacks are lists of saved commands with their order, execution time and values.

2.5.1.2 Data Proxy

The Data Proxy works as an alternative access point from MCS to telemetry sources. While nominal S/X-band operations access will be used with the NIS via SLE. Data Proxy is used to access the UHF chain, the FlatSat, the CAN bus directly (via TUG Adapter) and CORTEX

direct access for high speed CFDP uplink/experimenter access. Like the On-board Software, Data Proxy is itself an MO Services provider and can be commanded from SCOS.

2.6 FMS

FMS can transfer and synchronise data between the ground and the spacecraft. It can also create folders on the spacecraft and only synchronise specific files from the spacecraft to the ground and other way around.

3 AUTOMATION

Automation in this document describes the machinable execution of tasks given by a Mission Planning System or an operator for a pass outside of the working hours.
The automation concept is kept as simple as possible. MATIS provides a fixed platform with all functionality and tested procedures necessary to execute every expected task given to it.
Therefore, the automation consists out of two key elements. The first element is a static platform. It includes well tested procedures, activities and Groovy functions to be executed.

The second part is the Event Schedules (MAES) and User Schedules (MAUS) as an input from the Mission Planning. These files are automatically stored in a dedicated folder in MATIS. When a MAES is loaded into this folder it is automatically loaded into the MATIS Calendar with its time, ID and name. After the MAES is loaded the MAUS will be loaded into the Calendar as well and synchs with the existing events. Thus, the tasks of the MAUS have a dedicated starting time.
The MAUS is then waiting for the right events and pre-conditions to be satisfied to execute its task.
Tasks of the MAUS can execute procedures and activities.

In case of unexpected response from the spacecraft the operators are automatically informed by email from MATIS directly.

To summarise, each procedure used for automation is well tested and static, therefore normally does not change. This leads to a well-tested system able to be used as a flexible tool for operation due to the variable stackability of procedures in a MAUS.
Note: Customised non-standard procedures to fulfill the needs of a certain situation can always be added during working hours but should be tested on a flat-sat prior execution to/on the spacecraft.

The Mission Planning takes away the planning part of the automation. Therefore, automation is also possible without Mission Planning Software and can be prepared by the operator.

3.1 Input-Output-Flow

System	Input	Output
Pass Planning	TLE<-NORAD (orbit information)	Schedule File with AOS/LOS time
Mission Planning	Schedule File with AOS/LOS time	- Stack of commands (into SCOS) - MAES, MAUS (saved in special Repository in MATIS) - Experiment files for FBO
MATIS	MAES, MAUS	Commands to: - SCOS (through SMF) - SMF - SpaceShell - Scripts - External Programmes
SMF	Commands from MATIS	- Commands to SCOS - Emails - SMS
SCOS	Commands from MATIS through SCOS	- Log Files - Commands to Applications
FBO/FMS	Files for upload/download	- S/C: Files - Ground: Experimenter Files
Manual Stack	Commands from SCOS	Commands to S/C

3.2 Virtual Machines Setup

In operational mode two machine pairs are used for redundancy, called OSMATA & OSMCA for the first pair and OSMATB & OSMCB for the second virtual machine set. Both machines sets have the same setup and it can be switched between the chains in case of failure.
MATIS Server is running on OSMATx. SCOS, the NIS, the Prime Servers and SMF are running on OSMCx.

3.3 Key Elements for Automation

3.3.1 MAES

MAES are used to define all events and their date. These can be loaded into the calendar where they can be used by the MAUS, as described in the section below.
The MAES contain the event name, its ID and in “Constraints” the time of occurrence.

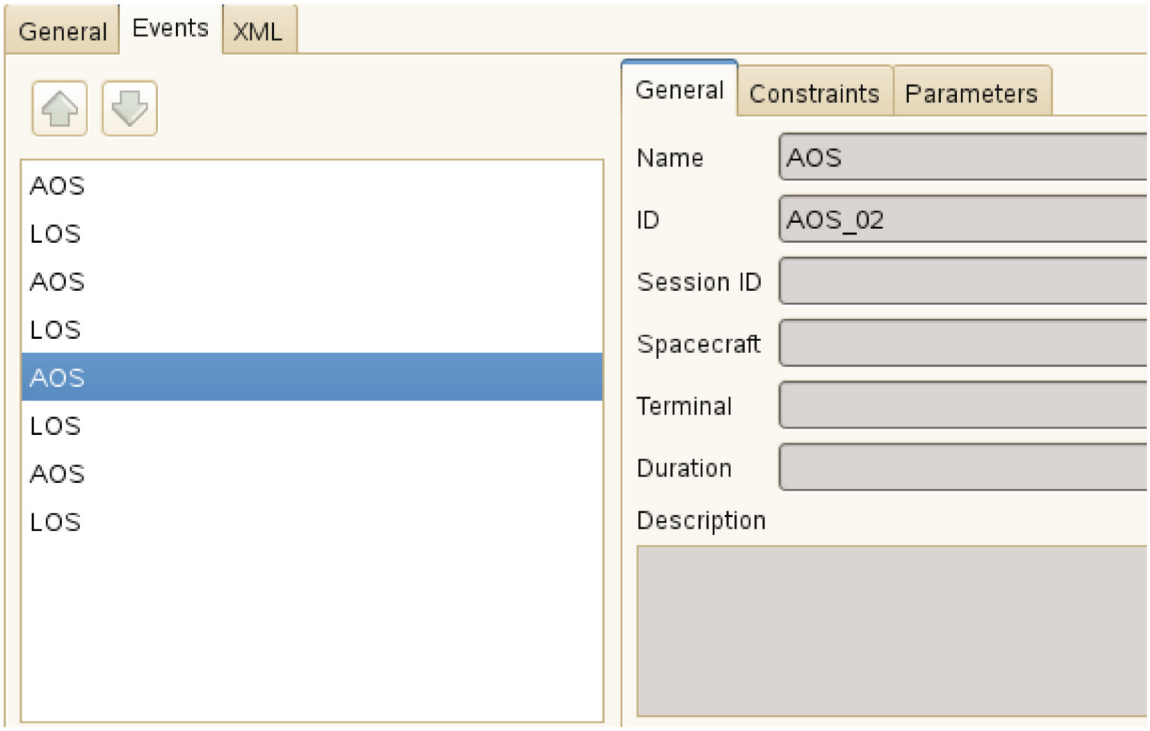


Figure 2: MAES in UI

The MAES is saved as an XML file. An example can be seen below.

General Events XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SCHEDULE_EVENT>
  <STATIC>
    <SCHED_EVENT_NAME>OS_TEST_MAES_SCHED_PASSES_00</SCHED_EVENT_NAME>
    <DESCRIPTION></DESCRIPTION>
    <SCHEMA_VERSION>1.1</SCHEMA_VERSION>
    <AUTHOR>Felix Hessinger</AUTHOR>
  </STATIC>
  <CONSTRAINTS>
    <TIMING>
      <REFERENCE_TIME>2019-02-28T17:04:21.147</REFERENCE_TIME>
    </TIMING>
  </CONSTRAINTS>
  <CONFIGURATION_CONTROL/>
  <PREDICTED_EVENT_LIST>
    <PREDICTED_EVENT>
      <NAME>AOS</NAME>
      <DESCRIPTION></DESCRIPTION>
      <INSTANCE_ID>AOS_00</INSTANCE_ID>
      <CONSTRAINTS>
        <TIMING>
          <RELATIVE>
            <EVENT_TIME>000:00:00:00.000</EVENT_TIME>
          </RELATIVE>
        </TIMING>
      </CONSTRAINTS>
      <SESSION_ID></SESSION_ID>
      <SPACECRAFT></SPACECRAFT>
      <TERMINAL></TERMINAL>
    </PREDICTED_EVENT>
```

Figure 3: MAES in XML

3.3.2 MAUS

The MAUS contains tasks which itself contain the dependency of a previous event, an event-precondition, a time limit for execution and a procedure to be executed.
When a certain task fulfills all pre-conditions then the procedure will be executed with the corresponding parameters.

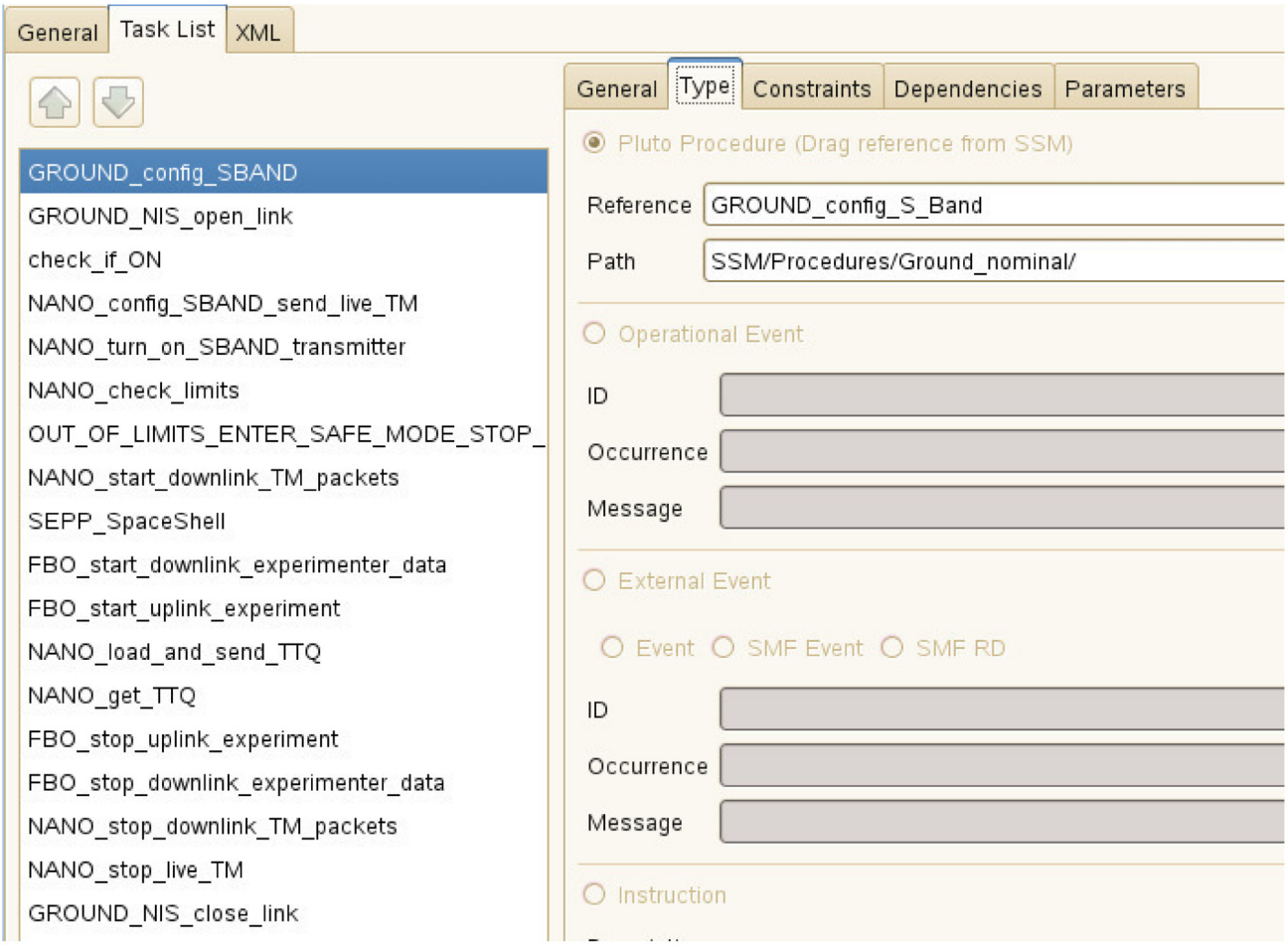


Figure 4: MAUS in UI

As the MAES beforehand, the MAUS is also saved in XML format.

3.3.3 Calendar and Global GANTT

The Calendar depicts the dependencies, timings and pre-conditions set in the MAUS tasks. It connects their dependencies and duration of maximum possible execution range. Additionally, the Calendar shows success and abort of the individual tasks in green and red respectively, giving a very fast overview to the operator of success or failure of autonomous passes.
An example for the Calendar view of one particular Schedule is depicted below.

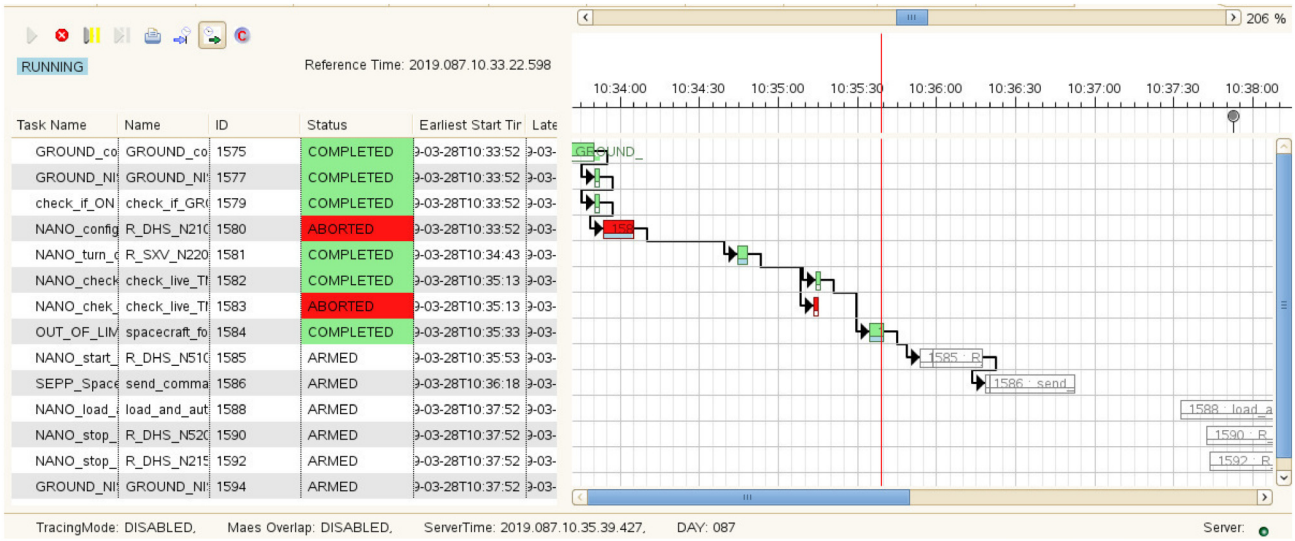


Figure 5: Example Calendar View of Specific Schedule

An additional useful tool is the Global GANTT. It shows all events and procedures to be executed. The names of the procedures are sorted alphabetically and can be seen on the left. Their maximum execution window is shown in the timeline on the right.
The corresponding image for the example above is depicted in the figure below.

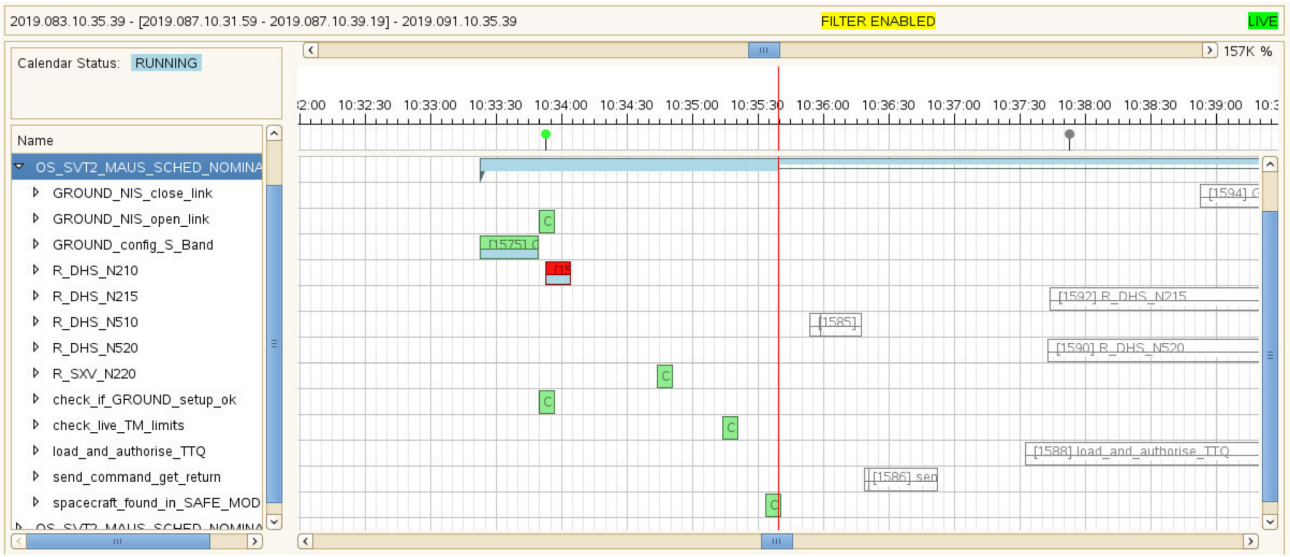


Figure 6: Example Global GANTT View of Specific Schedule

3.4 Elements and Timeline for a Pass Schedule

Each autonomous pass is represented in the MAES and MAUS file. Each file pair contains of six main blocks which are executed one after another:

- Configure (Open) Ground-Segment
- Configure(Open) Space-Segment
- Check Spacecraft Status
- Active “Interaction” with S/C
- Configure (Close) Space-Segment
- Configure (Close) Ground-Segment

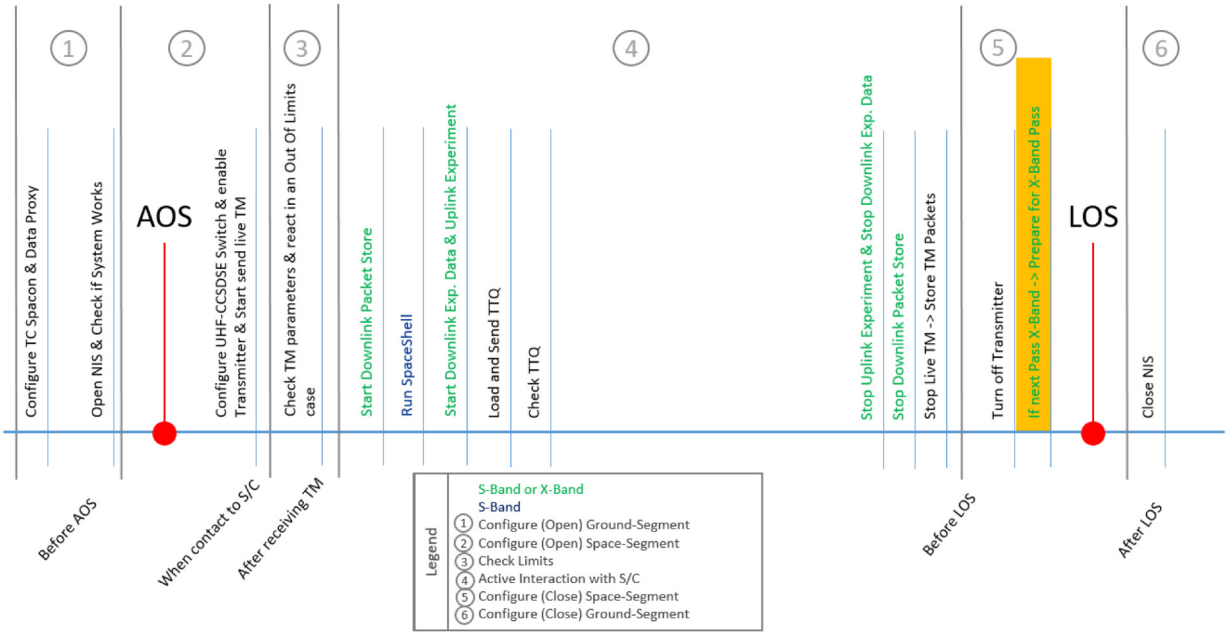


Figure 7: Timeline of a Pass Schedule in MATIS

4 QUICK-START MANUAL FOR MATIS AND AUTOMATION

For this quick-start it is important to follow the instructions step-by-step and in the correct order.

4.1 Start SCOS

On the correct machine, open the terminal and type “s2k.start”. After startup, log in, open the application manager and make sure all necessary servers are running. Additionally, check if the Data Proxy and the TC Spacon is setup in the correct configuration.
Servers to be running are divided into automation and prime servers.
Prime Servers:
- PARCfilerDist TM
- PARCfilerDist TC
- PARCfilerDist EV
- PARCretriever TM
- PARCretriever TC
- PARCretriever EV
- PARCmanager
- MISC
- USER
- EVENTS
- ACTIONS
- NCDU Admin
- FARC Server
- MULTI
- Verifier
- Releaser
- OBQM
- TPF
- PIF RT
- PIF PB
- CMD SUP Server
- Limits RT
- Limits PB
- Packetiser
- SCTL
- TCSM
- TMSM
- TCO
- TCH Model Live
- EVL Model Live
- OBE RTMod Live
- OBE PBMod Live
- TPH RTMod Live
- TPH PBMod Live

Automation (has to be started in this order):

- SMF SDS
- SMF SM
- SMF SRH

4.2 Start MATIS Server

On the correct machine , open the MATIS Server by starting “RunMatisServer.sh”. Wait until the terminal reports success, before starting MATIS Operation Manager.
(Currently the file is located at “/home/osops/MATIS/”)
It can also be started from SCOS in the automation tab directly.

4.3 Start MATIS Operation Manager

After MATIS Server startup, open the MATIS Operation Manager by starting “RunMatisOperationManager.sh”
(Currently the file is located at “/home/osops/MATIS/”)
When it opened, go to the SVN Repository view and click the button “New Repository Location” enter the URL https://sdereps.esa.int/svn/opssat-automation and enter your personal username and password to get access. Click Finish to download the repository to your local device. After the download has finished you will see a new repository popping up. Open it with the small arrow on the side, choose a branch or the trunk, right click on it and select “Check Out”.

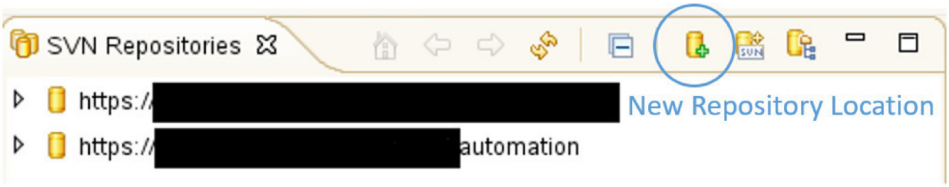


Figure 8: SVN Repositories View

After this the checked out the repository has to be selected as our working repository. Therefore, go to Window->Preferences, in there go to EGOS MATIS-> General->Repository and change the option Current Project to the project that has been checked out.

NOTE: The Operational Manager is not meant to edit procedures and to add new schedules etc. Therefore, it does not have the full functionality of the MATIS Designer. For bigger changes use the MATIS Designer.
In the MATIS Designer procedures, schedules etc. cannot be executed. When something is changed in the MATIS Designer then the changes have to be committed to the server, checked out or updated and executed in the MATIS Operation Manager.

4.4 Create Procedure

To create a procedure, right-click on a System Element (SE) in MATIS in the Repository view and select “New Activity Procedure”

4.5 Procedure Execution

The execution of a Procedure can mainly be done in two ways. By executing the Procedure directly or by assigning it to a MAUS.

4.5.1 Execute Procedure

To execute a Procedure double click or select the “Execute” option when right clicking on the specific Procedure.
A pop-up will be visible, here one could either enter a specific start time of the procedure or execute it directly.

4.5.2 Create MAUS and Assign Procedure to Task

Create a MAUS by right-clicking on a SE and selecting “New Schedule”. The naming convention can be found in [MAT-TRN-0001]. For OPS-SAT an example for a pass plan is OS_SVT2_MAUS_SCHED_NOMINAL.maus.
To assign a procedure add a task to the MAUS and drag and drop the procedure into the field “Reference” in the tab “Type” inside of the tab “Task List” in the MAUS.
If the procedure has parameters add them in the tab “Parameters”.
Constraints can also be set in the tab “Constraints”, such as earlies start time, event and stop time. Additionally, dependencies in “Dependencies” tab can be added to achieve a certain logic between tasks in this MAUS.
Additional information can be extracted from section 5.3.

4.6 Check Outcome

To check the outcome of a Procedure or Scheduled Task go to the “Realtime Log View”. A green color indicates success of the execution.
Note: Sometimes a red log is acceptable, but these cases have to be known or covert by a backup procedure that reacts to it (only possible with MAUS when dependencies are added, not with manually executed Procedure).

5 FAQ AND NON-NOMINAL ACTIONS

In this section a lot of common issues and non-nominal use cases that could arise are listed and its possible solution explained.
Please note that SCOS and SMF related issues are usually handled on the OSMCx machines and their configuration file system and MATIS configuration related tasks on the OSMATx machines.

5.1 MIB Update Implementation

There are several steps to update the MIB.
Step 1:
Get the .dat MIB files used by SCOS.
Step 2:
Open the "SVN Repository Exploring" view in MATIS Designer. Find and open the folder called "MIB".
Step 3:
Get the representative local workspace location of these files and copy paste the .dat files from the SCOS MIB.
Step 4:
Check out/update and then commit the files via the view "Project Explorer" in MATIS by right-clicking on the updated files -> Team -> Commit.

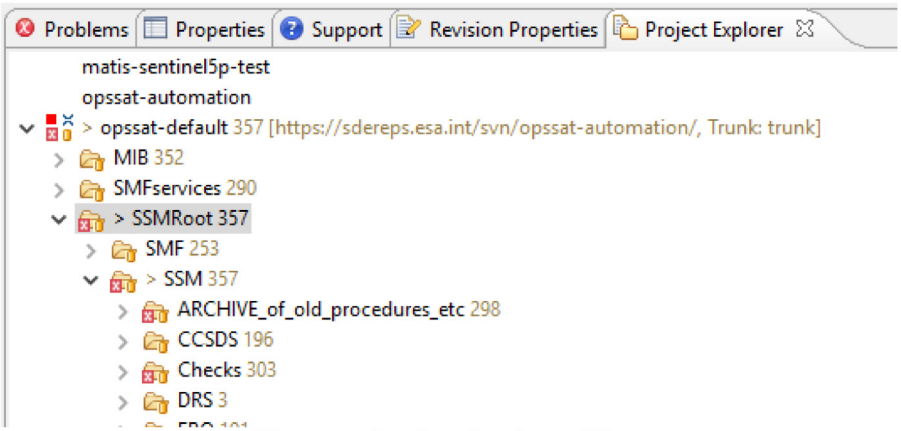


Figure 9: Project Explorer View

Step 5:
Check in MATIS Designer if the files have been updated.
(See number behind changed file. The number should now have increased behind every file that contains changes).
If no update has been applied. Check the files and continue with Step 1.
Step 6:
The MIB itself has now been updated on the MATIS side.
To get values/commands from the MIB go to "MATIS Modelling" tab in MATIS Designer and drag and drop the MIB content from "Spacecraft Data" to your repository.

If the Spacecraft Data is not up to date press the button with the green arrows in the upper right corner of this view. If it is still not up to date close and restart the MATIS Designer and check out the project again.

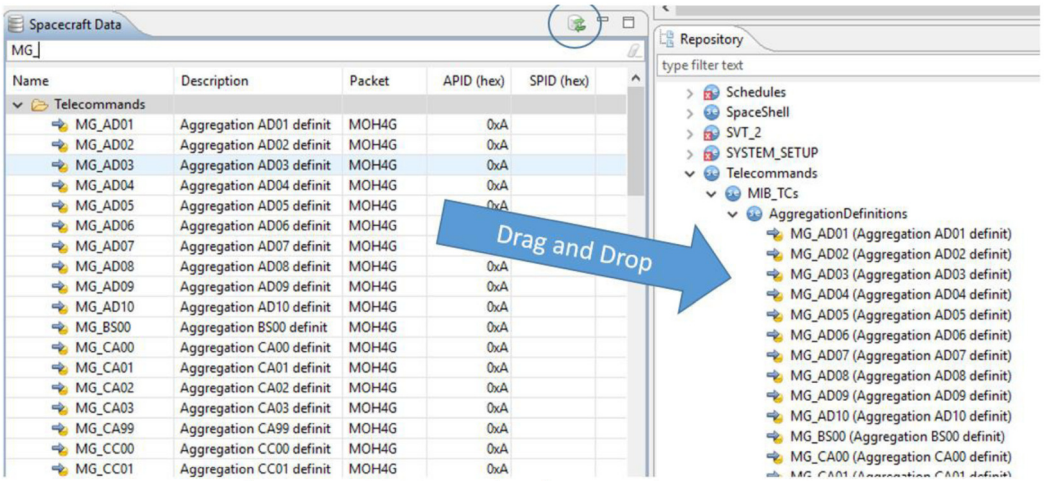


Figure 10: SCOS MIB implementation to MATIS

NOTE: To update the MISCconfig.dat file see 5.2

5.2 What are MISC variables and how to Update them?

The MISC variables contain custom variables which can be used by the user to synchronise between procedures as well as dynamic setup variables of SCOS and its peripherals.
E.g. CMD_TC_LINK which is used to verify if a connection to the satellite is enabled.
To do this several steps have to be executed.
Step 1:
Get any .dyn file in the “config” or “config_global” folder where the SCOS configuration files are stored. Check which parameters you need and add them to the MISCconfig.dat file in the MIB in MATIS.
The format in the MISCconfig.dat file is <ParameterName>TAB<Description>.
If all parameters should be updated from SCOS’s .dyn file to MATIS .dat file then use the python script “MATIS_MIB_MISC_dyn2dat_converter.py” which is located in folder Automation on Sharepoint.
Step 2:
After updating the file MISCconfig.dat either commit from MATIS Designer directly or use e.g. “SVN Tortoise” to commit the file to the MATIS SVN repository.
Step 3:
Check if the number next to the MISCconfig.dat file has been increased after a refresh of the repository view in MATIS Designer. If not continue with Step 1.

5.3 Edit MAUS or MAES

Open the Schedule you want to edit. Click into the appearing MAUS/MAES view, then press the pen “Unlock for Editing” in the upper left corner of the screen. Now you can edit the fields in the MAUS/MAES editor view.

To create/delete dependencies or to add/delete new tasks use the buttons on the left of the "Unlock for Editing" pen button.

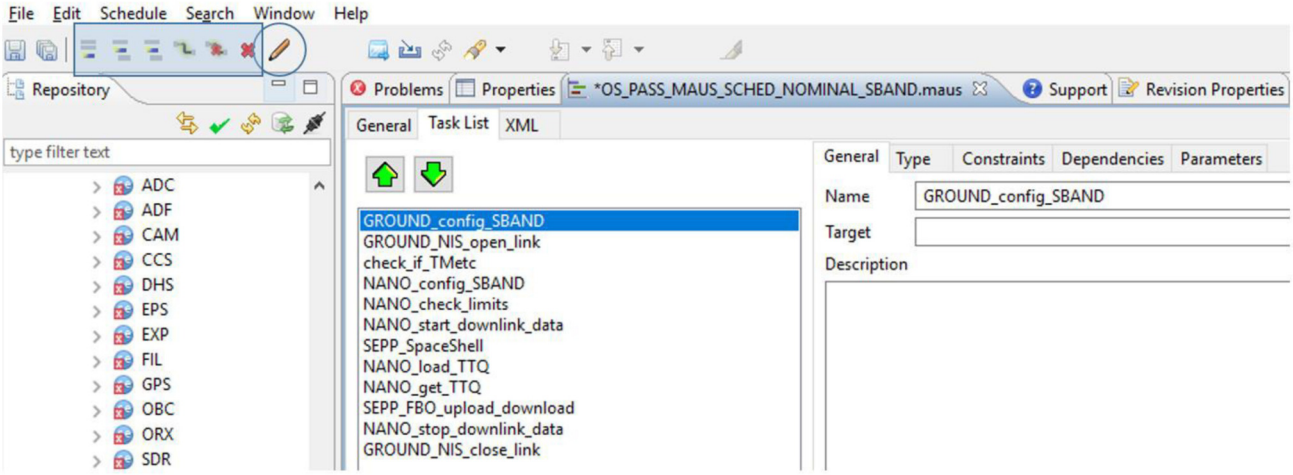


Figure 11: MAUS/MAES Editing Tools and General View

5.4 SCOS does not react when MATIS procedures are executed?

It might be that SCOS or the SMF Servers have been restarted and therefore, MATIS Server is still connected to some nonexistent SMF Servers.
Close the MATIS Operation Manager and the MATIS Server.
Make sure all other required servers are running.
Start the MATIS Server, wait for its completion and open MATIS Operation Manager afterwards.

5.5 Go to Operational Mode

In Operational Mode the connection to SVN is cut and only a snapshot/Tag of a state of the SVN Repository can be loaded at the Server. Additionally, only procedures which are in the state "validated" will be available.
Furthermore, the function of In-Tray execution of MAES and MAUS will be enabled.

Step 1:

Tag the current SVN Repository by File->Tag Current Repository… in the MATIS Designer.
The cleanup of the non-validated procedures is happening in this step.

Step 2:
Export Tagged Repository.

Step 3:
On the machine where MATIS Server is running open "MatisServerConfig.sta" and change the value of "esa.egos.matis.server.operational.mode.disabled" to "false".

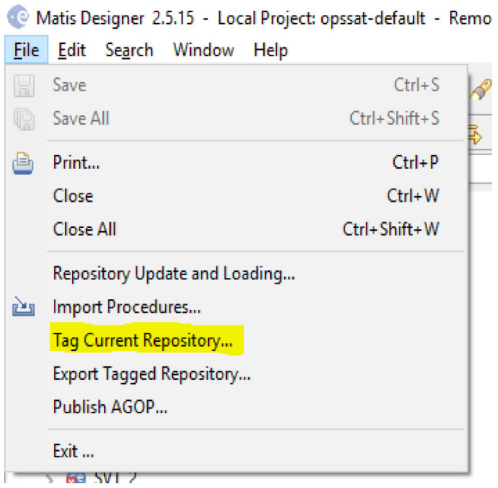


Figure 12: Tag Current Repository

Step 4:
Lay the exported Tagged Repository file into the dedicated MATIS Server folder

Step 5:
Restart MATIS Server.
(When opening MATIS Operation Manager the procedures might be locked. Instructions how to unlock it are given in an info box when trying to execute a procedure.)

5.6 Use IN-TRAY (Automatic Execution of MAES & MAUS)

IN-TRAY introduces a system where MAES and MAUS can be loaded into a specific folder and from this folder get executed automatically.

5.7 Adding a Driver on the example of the Calendar Driver

Check if the Calendar Driver is already installed by looking at the SMF SHR log. If a warning for the Calendar Driver is displayed as shown in the picture below, execute the following steps.

```
INFO SMF10301 System Element: TEST_MISSION.0.PRIME.SwissKnife.Tester.Execution  registered in the Naming S
WARNING SMF10103 Driver  -1 : esa.egos.smf.matis.drivers.calendarDriver.CalendarDriver NOT INITIALISED
CONFIG SMF400054 SDS registration system element success:  TEST_MISSION.SERVICE_REQUEST_HANDLER.TEST_FULL_
CONFIG SMF400036 SDS registration SMF au success:  SERVICE REQUEST HANDLER
```

Figure 13: Warning for not initialised Calendar Driver

Steps 1:
Go to ~/SMF/SMF/config/resources/xmlFiles/taskConfiguration/ and open the file SmfS2kSrhPrimeSingleDomainAllDrivers.xml.
In this file make sure the line
<Driver Name="esa.egos.smf.matis.drivers.calendarDriver.CalendarDriver"/> exists in the domain which should be used. In the example of OPS-SAT domain 0 is sufficient.

Note: If the location of the file differs take a look at the SRH log and search for "Configuration file:", the location of SmfS2kSrhPrimeSingleDomainAllDrivers.xml should be displayed in this line.

Step 2:
Check if the corresponding driver is in the SMF library in ~/SMF/SMF/lib/ with the name *matis.smf.<version>.jar*. If not existent add the corresponding file to the directory.

Step 3:
To align SMF with MATIS configuration types have to be generated on SMF. Therefore, first generate the types with:
>> cd ~/SMF/SMF/; source .pmakes
>> java -jar lib/smfTypeSchemaFileGenerator.jar -filePath ~/SMF/SMF/config/resources/schemaFiles/typesDefinition -fileName SmfTypes.xsd
After this place the file SmfMatisTestTypes.xsd (and SmfNisTypes.xsd if you want to use the NIS services –SLE) to ~/SMF/SMF/config/resources/schemaFiles/typesDefinition/.

The files can be found in
~/MATIS/current/Matis-Server/resources/config/SmfAccess/config/resources/schemaFiles/typesDefinition/

Step 4: (only needed for SLES11 and the Calendar Driver in particular)
The SMF libraries have to be deleted in ~/MATIS/Matis-Server/lib/.
Thus check this folder for files containing "smf-3.0.0*.jar" by the command
>>ls –la ~/MATIS/current/Matis-Server/smf .*
Whereas * is the typical placeholder for in Linux systems.
Remove the corresponding files with
>>rm -rf ~/MATIS/current/Matis-Server/lib/smf-3.0.0.jar*

Step 5 – Copy SMF libraries:
Copy the SMF libraries to ~/MATIS/current/Matis-Server/lib/
>> cp ~/SMF/SMF/lib/smfApi.jar ~/MATIS/current/Matis-Server/lib/smfApi.jar
>> cp ~/SMF/SMF/lib/smfCorePrimitiveTypes.jar ~/MATIS/current/Matis-Server/lib/ smfCorePrimitiveTypes.jar
>> cp ~/SMF/SMF/lib/smfGenericInterface.jar ~/MATIS/current/Matis-Server/lib/smfGenericInterface.jar
>> cp ~/SMF/SMF/lib/smfSystemElement.jar ~/MATIS/current/Matis-Server/lib/smfSystemElement.jar
>> cp ~/SMF/SMF/lib/smfUtil.jar ~/MATIS/current/Matis-Server/lib/smfUtil.jar

Note: Take care to put the files to the right machines, since both have the same repository structures, but SMF and MATIS-Server configuration is used by OSMCx and OSMATx, respectively.

Step 6: Restart SMF
Restart the SMF SDS, SMF SM and SMF SRH. During SMF SRF start-up see at the log and confirm that the driver was successfully initiated/started.

Step 7: Restart MATIS-Server
Restart the MATIS-Server to connect to the right SMF session. Additionally, restart the Operation Manager.
Check out the project again, merge the remote Automation Model to the local project by the button from the Repository-View. Commit all changes to the SVN-Repository.
To make sure all files are correctly synched, reload the MATIS-Server content from the central SVN repository.

Step 8: Update SMFservices .xml Files
When the driver is correctly installed, but the needed function is not shown inside of the "Ground Services", execute this step.

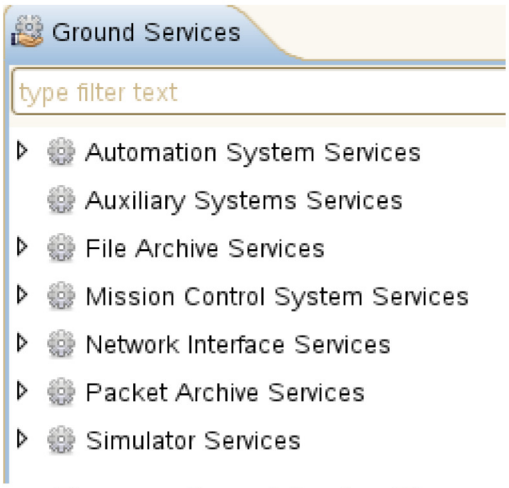


Figure 14: Ground Services View

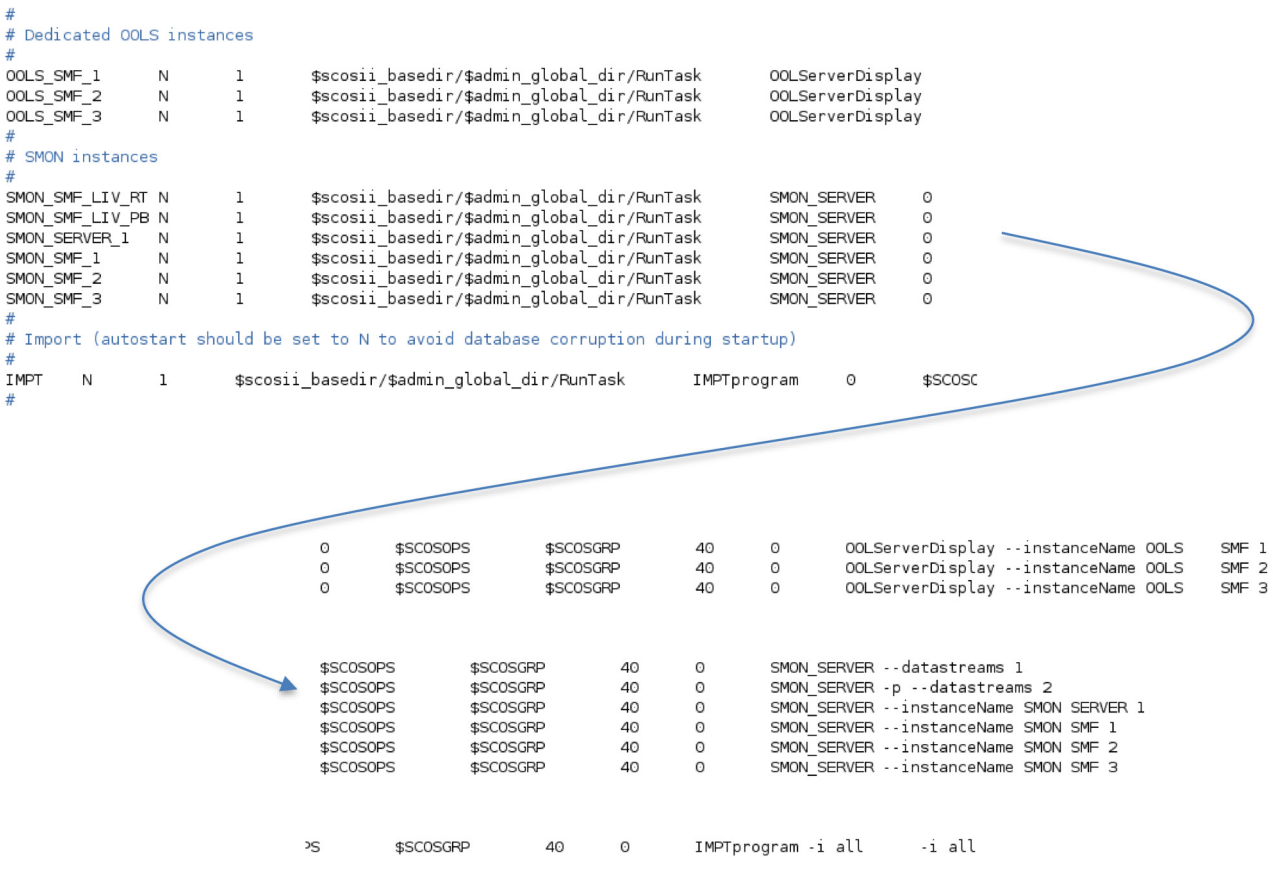
Copy the needed .xml file from
~/MATIS/current/Matis-Server/resources/config/SmfAccess/config/resources/xmlFiles/systemElementDefinition/
to
~/MATIS/workspace/Designer/opssat-default/SMFservices.
Then open the MATIS Designer, open the "Project Explorer" view and commit these files to the SVN Repository. Reload everything and the corresponding Ground Service will appear inside of the "Ground Services" view.
Now the System Element inside this Ground Service has to be dragged and dropped into the MATIS project from the "MATIS Modelling" tab. After this the functionality can be used.

5.8 Corba NS Driver NOT connected to Application Unit

As an example on this topic the case of not being able to subscribe to TM parameters of SCOS from MATIS. This was caused by a driver in SMF which was not connected to the corresponding Application Unit.
In ~/SMF/SMF_S2K/SmfS2kProperty.xml add the
"*SERVER_APPLICATION_UNIT_NAME"
"*SERVER_APPLICATION_UNIT_ARGUMENTS"
"*SERVER_APPLICATION_UNIT_MNEMONIC_NAME"
Example:

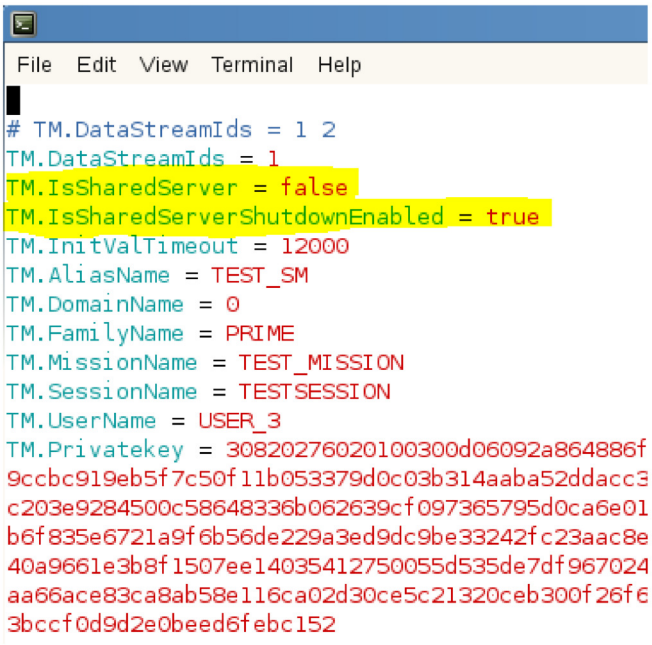
```
<entry key="SHARED_TM_SERVER_APPLIATION_UNIT_NAME">SMON_SERVER_1</entry>
<entry key="SHARED_TM_SERVER_APPLIATION_UNIT_ARGUMENTS">1</entry>
<entry
key="SHARED_TM_SERVER_APPLIATION_UNIT_MNEMONIC_NAME">SMON_SERVER_1</entry>
```

The MNEMONIC_NAME can be taken from the first entry of a row inside of ~/admin/TKMAconfigFile.txt
If the service does not yet exist, create a new entry/line with the corresponding values.



Specifically for this case an additional change has been made, but this step is not necessary for general Application Unit connection.

In ~/MATIS/current/Matis-Server/resources/config/MatisServerConfigAbstractionLayerAccess.properties the lines
TM.IsSharedServer = false
TM.IsSharedServerShutdownEnabled = true
have to be added.



5.9 Max Number of Sessions Reached

Each MATIS service needs a session, usually this number is lower than 5, but for OPS-SAT this number sometimes exceeds this value.
The MATIS default configuration sets the variable to MAX_NUMBER_SESSIONS to 5. Nevertheless, it might be that more than 5 sessions are needed and therefore, this configuration should be changed.
In the file ~/SMF/SMF/SmfProperty.xml change the number for the value MAX_NUMBER_SESSIONS to a high number e.g. 200, save and restart the SMF and the MATIS-Server.

B Useful Links for ESA Internals

OPS-GD OPS-SAT Wiki:
http://gdwiki.esoc.esa.int/wiki/OPS-SAT
SCOS User Manual:
file://esaad/esoc/OPS-GI-CurrentRef/SCOS-2000/SCOS-2000%20R5.4.21/Documentation/S2K/sum.html
Uberlog:
http://uberlog1.esoc.esa.int
Redmine:
https://opssat1.esoc.esa.int
Sharepoint:
https://esateamsite.sso.esa.int/DOPS/O/S/A/OPS-SAT
Gitlab:
https://gitlab.esa.int/OPS-SAT
vSphere:
https://smile2.esoc.esa.int/
ESA connect:
https://one.esa.int/home/
NMF information:
https://dmarszk.github.io/MOWebViewer4NMF/?u=Platform/AutonomousADCS/Data/AttitudeTelemetry
https://opssat1.esoc.esa.int/projects/experimenter-information/dmsf?folder_id=6
Get Paramters defined in Aggregation:
https://gitlab.esa.int/OPS-SAT/opssat_mcs/blob/development/SPACE_SEGMENT/simulator/OPS_SAT/OPSSAT_MO_MIB/src/main/resources/Aggregations.xml

DOCUMENTATION:
All Documentation:
//esaad/esoc/OPS-GI-CurrentRef
MATIS:
http://10.32.58.189/
Latest MATIS release:
http://10.32.58.189/release2.5.15/index.html
SCOS:
file://esaad/esoc/OPS-GI-CurrentRef/SCOS-2000/SCOS-2000%20R5.4.21/Documentation/S2K/index.html

NMF:
https://dmarszk.github.io/MOWebViewer4NMF/?u=Platform/AutonomousADCS/Data/AttitudeTelemetry
AGGREGATIONS:
https://gitlab.esa.int/OPS-SAT/opssat_mcs/blob/development/SPACE_SEGMENT/simulator/OPS_SAT/OPSSAT_MO_MIB/src/main/resources/Aggregations.xml
MISSION-WIKI:
http://gdwiki.esoc.esa.int/wiki/OPSSAT-INSTALL-MATIS#MATIS_server_fails_to_start_due_to_.22Server_SSL_certificate_untrusted.22

C SCOS Documentation Files Overview

Architecture Design Documents

ADD-0001 - System Level
ADD-0002 - Commanding
ADD-0003 - Monitoring
ADD-0004 - Archiving
ADD-0005 - Mimic
ADD-0006 - Event and Actions
ADD-0007 - Operation Language
ADD-0011 - OBSM
ADD-0012 - Users, Roles & Privilegies
ADD-0013 - Utilities
ADD-0014 - Mechanisms
ADD-0016 - Task Launcher
ADD-0017 - TM Packetiser and Packet Displays
ADD-0018 - Client Packet Distributor
ADD-0019 - MISC
ADD-0020 - DDD Generation Tool
ADD-0022 - System Control
ADD-0023 - Desktop
ADD-0024 - Packet Management
ADD-0025 - System Configuration Manager
ADD-0027 - System Configuration Monitor
ADD-0028 - Database GUI
ADD-3000 - EXIF and WEBRM
ADD-4000 - DDS
ADD-4003 - MCCM Test Tool
ADD-4004 - DB Tool
ADD-5000 - Remote Desktop
ADD-5002 - Lightweight Framework
ADD-0001 - CHAPP
SDD-1001 - PARC
SDD-1002 - Data Provision Services
SDD-1003 - TM Snapshot Checker Server
SDD-1004 - PUS Display Model
SDD-0001 - EUD-SMH
SDD-0001 - Command Supervisor

Detailed Design Documents

DDD-0001 - PARC MySQL Driver
DDD-0002 - PARC Oracle Driver
DDD-0003 - PARC RAPID Driver
DDD-0004 - PARC Manager
DDD-0005 - PARC Utilities
DDD-0006 - PARC Filer & Distributor
DDD-0007 - PARC Retriever
DDD-0008 - PARC Replicator
DDD-0009 - PARC Proxy
DDD-0010 - PARC DBM Driver
DDD-0001 - CHAPP

Figure 2: SCOS documentation overview of software design documents [25].

Software User Manuals

SUM-0003 - Getting Started
SUM-0004 - Time & Dates
SUM-0005 - MISC
SUM-0007 - MSG
SUM-0008 - PDS
SUM-0010 - SIG
SUM-0015 - MVC
SUM-0016 - Values
SUM-0018 - MIB Applications (POST++)
SUM-0019 - Synthetic Parameters
SUM-0026 - Event & Actions
SUM-0030 - TM Packetiser
SUM-0031 - CPD
SUM-2000 - Top Level
SUM-2110 - General Desktop
SUM-2111 - Task Launcher
SUM-2120 - Event Log Display
SUM-2130 - Message Watch Tool
SUM-2140 - PDS Administration Tool
SUM-2210 - Telemetry Desktop
SUM-2220 - Out-of-Limits Display
SUM-2230 - Variable Packet Display
SUM-2232 - TM Packet History
SUM-2233 - OnBoard Event History
SUM-2240 - TM SPACON
SUM-2250 - TM Printouts
SUM-2260 - Graphic Editor
SUM-2270 - MIMIC Editor
SUM-2290 - TM Configuration Checks
SUM-2310 - Manual Stack
SUM-2330 - TC SPACON
SUM-2340 - OnBoard Queue Model Display
SUM-2350 - Auto Stack
SUM-2360 - TC History Display
SUM-2400 - OBSM
SUM-3005 - EXIF Test Client
SUM-7001 - DDS Test Tool
SUM-7020 - Generic Packetiser Test Tool
SUM-7021 - Database Management Tool
SUM-7022 - Telemetry Replayer
SUM-7023 - TMPI (Telemetry Packet Injector)
SUM-7024 - TC Tool
SUM-7025 - Test Database
SUM-7026 - System Configuration Manager
SUM-0001 - Automated Regression Testing
SUM-0001 - CHAPP
SUM-1001 - EUD-SMH
SUM-1002 - PARC (Operational)
SUM-5000 - Remote Desktop
SUM - Command Supervisor

Figure 3: SCOS documentation overview of software user manuals [25].

Interface Control Documents

ICD-0001 - Database Import
ICD-0002 - Stack Import
ICD-0003 - Task Parameter File
ICD-0004 - Telemetry Packet
ICD-0005 - Command Injection
ICD-0006 - TM Parameter Data Provision Services (CORBA)
ICD-0007 - Event Injection
ICD-0008 - Command History Data Provision Services
ICD-0009 - Event History Data Provision Services
ICD-0010 - CORBA-based External Interface Services
ICD-0011 - Parameter Injection
ICD-0012 - PIF Server [a]
ICD-0012 - PIF Server [b]
ICD-0014 - OBSM External Interface
ICD-0015 - Packetiser
ICD-0016 - Command Source Handler
ICD-0020 - CORBA External Interface for Data Provisions
ICD-0022 - TM Packet Data Provision Services
ICD-0023 - CPD
ICD-0024 - Archive Subsystem
ICD-0027 - External Interface MIB Data Injection Services
ICD-0028 - Telemetry Replayer
ICD-0029 - TM Packet Injector
ICD-0030 - TC Tool
ICD-0031 - Command Source
ICD-0040 - TM SPACON
ICD-0041 - TC SPACON
ICD-0042 - TC History
ICD-0043 - On Board Queue Display
ICD-0044 - TM Packet History Display
ICD-0045 - On Board Event Handler
ICD-0046 - On Board Software Maintenance
ICD-0047 - Event Log Display
ICD-0048 - Common GUI Separation
ICD-0050 - TC Tool Model-View
ICD-0051 - TC Sequence Definitions
ICD-0055 - PDS pkt Source
ICD-1002 - PARC Internal Interfaces
ICD-1003 - PARC External Interfaces
ICD-1005 - Rapid Library
ICD-1006 - Data Provision Services
ICD-1008 - TM Packet Injector Server
ICD-1009 - PUS Display Model
ICD-5001 - TM Display & Printouts Definitions Service
ICD-5002 - TM Model C++
ICD-5003 - TM Processing CORBA Service
ICD-5004 - OL Model C++
ICD-5006 - TM IDL General Data Types
ICD-0001 - CHAPP
ICD-0001 - Command Supervisor

Figure 4: SCOS documentation overview of interface control documents [25].

D OPS-SAT MATIS Test Log Matrix

This Appendix contains a matrix of tests performed on the system. These tests range from SCOS commands to MATIS procedures and SMF functionality. Table 1 gives a list of abbreviations for the ID-names used in the test matrix. The ID is a combination of the most dominant system responsible for the corresponding test, and the number of this systems occurence inside this matrix.

Table 1: Abbreviations used for ID.

Abbreviation	Explanation
SpS	SpaceShell
CMD	Command in SCOS
SCOS	SCOS
MAT	MATIS
MAUS	MAUS
MAES	MAES
PRO	Procedure/Activity

ID	DATE	ACTIVITY/PROCEDURE/SCHEDULE ID	COMMAND ID*	DESCRIPTION	VERSIONS				SUCCESS	FAILURE	WARNING
					MATIS	PRO	MIB	OBSW			
CMD-0001	2019-03-01		M4B1707b	OPS enableCriticalCommand	2.5.14		1.8	v2.20.0	Y		
CMD-0002	2019-03-01		M4B1401s	OPS alive	2.5.14		1.8	v2.20.0	Y		
MAT-0001	2019-03-14			MATIS: Passing of Parameters from MAUS to Procedure (Boolean, String, Unsigned Integer)	2.5.14		1.8	v2.20.0	Y		
MAT-0002	2019-03-14			MATIS: Insertion of events (MAES) and correct loading of MAUS	2.5.14		1.8	v2.20.0	Y		Y
SpS-0001	2019-03-14			MATIS/SpaceShell: executing SpaceShell from MATIS with different String inputs	2.5.14		1.8	v2.20.0		Y	Y
SpS-0002	2019-03-15	get_cpu_load		MATIS/SpaceShell: executing SpaceShell from MATIS with different String inputs	2.5.14		1.8	v2.20.0	Y	Y	Y
SpS-0003	2019-03-15	get_memory_available		MATIS/SpaceShell: executing SpaceShell from MATIS with different String inputs	2.5.14		1.8	v2.20.0	Y	Y	Y
SpS-0004	2019-03-15	get_SEPP_uptime		MATIS/SpaceShell: executing SpaceShell from MATIS with different String inputs	2.5.14		1.8	v2.20.0	Y	Y	Y
SpS-0005	2019-03-15	send_command_get_return		MATIS/SpaceShell: executing SpaceShell from MATIS with different String inputs	2.5.14		1.8	v2.20.0	Y	Y	Y
MAT-0003	2019-03-21	SetServiceMode		MATIS/SMF/SCOS: TC SPACON Service Mode change between "CLTU" and "PACKET"	2.5.14		1.8	v2.20.0	Y		
CMD-0003	2019-03-21		G2A0101s	MATIS/DATA PROXY: change modes in data proxy	2.5.14		1.8	v2.20.0		Y	Y
MAT-0004	2019-03-21	LoadAndAuthoriseSavedStack		Load and execute saved SCOS command stack	2.5.14		1.8	v2.20.0		Y	
CMD-0004	2019-03-22		G2A0101s	MATIS/DATA PROXY: change modes in data proxy	2.5.14		1.8	v2.20.0	Y	Y	Y
SCOS-0001	2019-03-22	any activity/procedure	any command	TC UPLINK VERIFICATION (UV) has "UV Fail" status -> no commands can be send anymore	2.5.14		1.8	v2.21.0			
MAT-0005	2019-03-25	StopCalendar		Stops the caledar and all planned schedules	2.5.14		1.8	v2.21.0		Y	

Figure 5: Detailed pass events for phase "Configure (Close) Ground-Segment" (1a).

	ID	DATE	NOTES	SOLVED
	CMD-0001	2019-03-01	no problems found yet	
	CMD-0002	2019-03-01	no problems found yet	
	MAT-0001	2019-03-14	no problems found yet	
	MAT-0002	2019-03-14	MAES has to be inserted first, then MAUS. If a task should be executed after an other task that has event dependency, the same event dependency has to be added to the task or MATIS will insert incorrectly.	
	SpS-0001	2019-03-14	If SpaceShell command has no Spaces then the command will be executed correctly. If a it has spaces between the words then it is interpreted as several inputs and therefore, executing incorrectly!	Has been solved at 15/03/2019
	SpS-0002	2019-03-15	Groovy function "delete_first_lines" has to be updated to cut printed lines to get only wanted output	Retested at SpS-006
	SpS-0003	2019-03-15	Groovy function "delete_first_lines" has to be updated to cut printed lines to get only wanted output	Retested at SpS-006
	SpS-0004	2019-03-15	Groovy function "delete_first_lines" has to be updated to cut printed lines to get only wanted output	Retested at SpS-006
	SpS-0005	2019-03-15	Groovy function "delete_first_lines" has to be updated to cut printed lines to get only wanted output	Retested at SpS-006
	MAT-0003	2019-03-21	no problems found yet	
	CMD-0003	2019-03-21	switch from EGSE1 to OSUBBE worked once, other way around was not possible. Additionally, the same procedure from EGSE1 to OSUBBE that worked beforehand does not work anymore when executed again.	Workaround: go to PACKET mode in TC SPACON then change mode in DATA PROXY then go to the Service mode in TC SPACON needed for following operation. see CMD-0004
	MAT-0004	2019-03-21	Does not work. Error: Failure of Service (maybe problem with a driver -> ask support) Jira entry has been created (AUTOMATI-217) (https://sdejira.esa.int/browse/AUTOMATI-217)	solved in MAT-0017
	CMD-0004	2019-03-22	During the change in DATA PROXY the Service mode in TC SPACON has to be in PACKET mode!! (Service mode can be changed after DATA PROXY change.	see CMD-0006
	SCOS-0001	2019-03-22	This might happen when no sending verification/feedback is received.	SCOS: Data proxy: in tab "Release/PTV/CEV Configuration" press Reset button of "UV Status" MATIS: use "ResetGlobalUv of TcMCConfiguration"
	MAT-0005	2019-03-25	WARNING SMF10103 Driver -1 : esa.egos.smf.matis.driver/calendarDriver NOT INITIALISED	

Figure 6: Detailed pass events for phase "Configure (Close) Ground-Segment" (1b).

						VERSIONS						
ID	DATE	ACTIVITY/PROCEDURE/SCHEDULE ID	COMMAND ID*	DESCRIPTION	MATIS	PRO	MIB	OBSW	SUCCESS	FAILURE	WARNING
MAT-0006	2019-03-25	rtTmFlow of status of...@reference		gets the TM flow status	2.5.14		1.8	v2.21.0		Y	
CMD-0005	2019-03-25	config_UHF_CCSDS_swtich	MAx1302	Switches TM flow to flow over UHF or CCSDS engine	2.5.14		1.8	v2.21.0	Y		Y
MAT-0007	2019-03-25			Executing MAES to be inserted into the Calendar	2.5.14		1.8	v2.21.0	Y		
SpS-0006	2019-03-25	get_cpu_load, get_memory_available, get_SEPP_uptime, send_command_get_return		Retesting of SpS-002 to SpS-005	2.5.14		1.8	v2.21.0	Y		Y
MAT-0008	2019-03-26	SwissKnife: Send of Email of...		Sends an email to the operators	2.5.14		1.8	v2.21.0		Y	
MAT-0009	2019-03-26	with directives dynamic_ptv := "overridden" end with;		sets the command option dynamit_ptv to overridden which enables the command to be send without having TM FLOW	2.5.14		1.8	v2.21.0		Y	
MAT-0010	2019-03-27	Open of SleControl of....		Opens the link to NIS	2.5.14		1.8	v2.21.0		Y	
MAT-0011	2019-03-27	openTMlink of NIS		Opens the link to NIS	2.5.14		1.8	v2.21.0	Y	Y	Y
MAT-0012	2019-03-28	OS_SVT2_MAUS_SCHEDULE_NOMINAL_SBAND_1		Schedule test for combined (time and event based) execution of tasks and their procedures	2.5.14		1.8	v2.21.0	Y		
MAT-0013	2019-03-28	OS_SVT2_MAUS_SCHEDULE_NOMINAL_SBAND		Schedule test for event based execution of tasks and their procedures	2.5.14		1.8	v2.21.0	Y		Y
MAT-0014	2019-03-29	openTMlink of NIS		Opens the TM link to NIS	2.5.14		1.8	v2.21.0	Y		Y
MAT-0015	2019-03-29	closeTMlink of NIS		Closes the TM link to NIS	2.5.14		1.8	v2.21.0	Y		
MAT-0016	2019-03-29	SwissKnife: Send of Email of...		Sends an email to the operators	2.5.14		1.8	v2.21.0	Y		
MAT-0017	2019-03-29	LoadAndAuthoriseSavedStack		Load and execute saved SCOS command stack	2.5.14		1.8	v2.21.0	Y		
MAT-0018	2019-03-29	LoadSavedStack		Load saved SCOS command stack	2.5.14		1.8	v2.21.0	Y		
MAT-0019	2019-04-01	Reset_FPGA of SYSTEM_SETUP		Resets the FPGA running on machine OSEGSE	2.5.14		1.8	v2.21.0	Y		Y

Figure 7: Detailed pass events for phase "Configure (Close) Ground-Segment" (2a).

ID	DATE	NOTES	SOLVED
MAT-0006	2019-03-25	Does not get any return. SHOULD be TM_FLOW or NO_TM_FLOW -> Misc variable shall be created transfering the value of existing variable CMD_TM_FLOW in SCOS to MATIS	MISC variable CMD_FLOW_LINK is substituting this command
CMD-0005	2019-03-25	Works when TM FLOW, if no TM FLOW dynamix_ptv := "overridden" has to be used	
MAT-0007	2019-03-25	no problems found yet	
SpS-0006	2019-03-25	SpaceShell: no problems found yet EGSE: Known problem. Reported in Jira: OPSSAT-227 https://sdejira.esa.int/browse/OPSSAT-227?jql=project%20%3D%20OPSSAT%20AND%20text%20~%20EGSE	
MAT-0008	2019-03-26	Access to server is denied due to Firewall. "Sending the email to the following server failed: xxxxxx" Possible solutions: - Firewall exeption for existing address - SwissKnife update with new address and email address	solved in MAT-0016 (Firewall has been configured to enable sending emails from smtps-int.esa.int:465)
MAT-0009	2019-03-26	Does not set option dynamic_ptv in command to overridden. Jira entry for SCOS exists (S2K-8517) (https://sdejira.esa.int/browse/S2K-8517) Jira entry has been created (MATIS-242) (https://sdejira.esa.int/browse/MATIS-242) **Workaround:** - Use "LoadAndAuthoriseCommand of CommandInjection_BD" to set option -	
MAT-0010	2019-03-27	SMF driver SleControl is not working/installed. - installation in progress	solved in MAT-014
MAT-0011	2019-03-27	Drivers have been installed and NIS can be opened. Update: Same configuration does not work anymore.	solved in MAT-017
MAT-0012	2019-03-28	no problems found yet	
MAT-0013	2019-03-28	If several tasks have AOS as earliest start condition and LOS as latest stop condition and are dependent on each others completion, then when a task does not execute, all following tasks can also not execute (due to logic).	
MAT-0014	2019-03-29	no problems found yet (opens as expected, but does not yet connect due to missing configuration)	
MAT-0015	2019-03-29	no problems found yet	
MAT-0016	2019-03-29	no problems found yet (For security/safety: Emails can only be sent to internal people with email ending with @esa.int)	
MAT-0017	2019-03-29	no problems found yet	
MAT-0018	2019-03-29	no problems found yet	
MAT-0019	2019-04-01	no problems found yet (If it does not work check key of machines for authorisation via ssh command)	

Figure 8: Detailed pass events for phase "Configure (Close) Ground-Segment" (2b).

ID	DATE	ACTIVITY/PROCEDURE/SCHEDULE ID	COMMAND ID*	DESCRIPTION	VERSIONS				SUCCESS	FAILURE	WARNING
					MATIS	PRO	MIB	OBSW			
MAT-0020	2019-04-01	openTClink of NIS		Opens the TC link to NIS	2.5.14		1.8	v2.21.0	Y		Y
MAT-0021	2019-04-01	closeTClink of NIS		Closes the TC link to NIS	2.5.14		1.8	v2.21.0	Y		Y
CMD-0006	2019-04-01		G2A0101s	DATA PROXY: change modes in data proxy	2.5.14		1.8	v2.21.0		Y	
PRO-0001	2019-04-01	R_DHS_N210		switches NANO TM between UHF and CCSDSE + sets TM flow or storage	2.5.14		1.8	v2.21.0		Y	Y
PRO-0002	2019-04-01	R_DHS_N210		switches NANO TM between UHF and CCSDSE + sets TM flow or storage	2.5.14		1.8	v2.21.0	Y		Y
SpS-0007	2019-04-03	get_cpu_load		gets cpu load of SEPP	2.5.14		1.8	v2.21.0	Y		
SpS-0008	2019-04-03	get_memory_available		gets memory available	2.5.14		1.8	v2.21.0	Y		
SpS-0009	2019-04-03	get_SEPP_uptime		gets SEPP uptime	2.5.14		1.8	v2.21.0	Y		
SpS-0010	2019-04-03	send_command_get_return		sends any command and returns the output	2.5.14		1.8	v2.21.0	Y		
MAUS-0001	2019-04-03	OS_SVT2_MAUS_SCHED_NOMINAL_S BAND		skeleton of a SBAND pass (test of system)	2.5.14		1.8	v2.21.0	Y		Y
MAUS-0002	2019-04-03	OS_SVT2_MAUS_SCHED_NOMINAL_ UHF		skeleton of a UHF pass (test of system)	2.5.14		1.8	v2.21.0	Y		Y
MAES-0001	2019-04-03			creates events for two fake passes	2.5.14		1.8	v2.21.0	Y		
PRO-0003	2019-04-04	OBC_set_current_time		sets current time as OBC time	2.5.14		1.8	v2.21.0	Y		
CMD-0007	2019-04-04		M4B1707b	enable and disable critical commands	2.5.14		1.8	v2.21.0	Y		
CMD-0008	2019-04-04		M4B1301b	enable/disable OPS useAutomaticTime	2.5.14		1.8	v2.21.0	Y		
CMD-0009	2019-04-04		M4B1304b	sets the OBC time	2.5.14		1.8	v2.21.0	Y		
SCOS-0002	2019-04-05	Start of TaskManager		starts applications in SCOS	2.5.14		1.8	v2.21.0	Y		Y

Figure 9: Detailed pass events for phase "Configure (Close) Ground-Segment" (3a).

ID	DATE	NOTES	SOLVED
MAT-0020	2019-04-01	no problems found yet (opens as expected, but does not yet connect due to missing configuration)	
MAT-0021	2019-04-01	if executed fast after the openTClink procedure then it might not close the link and waits until timeout without closing the link itself	
CMD-0006	2019-04-01	Changing the Data Proxy profile to the same profile that is already activated De-Activates the data proxy -> cannot be re-activated by the command G2A0101s itself, but has to be started manually. JIRA entry has been created (OPSSAT-230) (https://sdejira.esa.int/browse/OPSSAT-230)	Has been solved in CMD-0009
PRO-0001	2019-04-01	with current version of MATIS this is not possible due to missing functionality/bug in its software. -> will be fixed in next update. **Workaround:** - use LoadAndAuthoriseCommand to pass it as an array	workaround applied in PRO-0002
PRO-0002	2019-04-01	Workaround has been used, now it is working When MATIS is updated then procedure should be changed to correct format without workaround. **Workaround:** - use LoadAndAuthoriseCommand to pass it as an array	
SpS-0007	2019-04-03	Had to be retested due to changes of the port inside of the SpaceShell and due to updates in the procedure itself	
SpS-0008	2019-04-03	Had to be retested due to changes of the port inside of the SpaceShell and due to updates in the procedure itself	
SpS-0009	2019-04-03	Had to be retested due to changes of the port inside of the SpaceShell and due to updates in the procedure itself	
SpS-0010	2019-04-03	Had to be retested due to changes of the port inside of the SpaceShell and due to updates in the procedure itself	
MAUS-0001	2019-04-03	Careful with the parts of the NIS. These are not yet implemented and not working yet. The FBO procedures are currently dummy procedures and will only create a log message, but have no function	
MAUS-0002	2019-04-03	NIS is not yet working with UHF	
MAES-0001	2019-04-03	no problems found yet	
PRO-0003	2019-04-04	no problems found yet	
CMD-0007	2019-04-04	no problems found yet	
CMD-0008	2019-04-04	no problems found yet	
CMD-0009	2019-04-04	no problems found yet	
SCOS-0002	2019-04-05	The applications that should be started has to exist in SCTLconfig.xml If configured correctly can be seen when the EUD-SMH is opened and SCTL View is active. (full path will be displayed)	

Figure 10: Detailed pass events for phase "Configure (Close) Ground-Segment" (3b).

						VERSIONS							
ID	DATE	ACTIVITY/PROCEDURE/SCHEDULE ID	COMMAND ID*	DESCRIPTION	MATIS	PRO	MIB	OBSW	SUCCESS	FAILURE	WARNING
SCOS-0003	2019-04-05	Stop of TaskManager		stop applications in SCOS	2.5.14		1.8	v2.21.0	Y		Y
PRO-0004	2019-04-05	start_Application_in_SCOS		starts application in SCOS by passing variable arguments	2.5.14		1.8	v2.21.0	Y		Y
PRO-0005	2019-04-05	stop_Application_in_SCOS		stops application in SCOS by passing variable arguments	2.5.14		1.8	v2.21.0	Y		Y
CMD-0008	2019-04-09		G2A0102s	configures data proxy to idle mode	2.5.14		1.8	v2.21.0	Y		Y
CMD-0009	2019-04-09		G2A0101s	DATA PROXY: change modes in data proxy	2.5.14		1.8	v2.21.0	Y		
MAT-0022	2019-04-09	Start of TaskManager (data-proxy)		starts data proxy	2.5.14		1.8	v2.21.0	Y		Y
MAT-0023	2019-04-09	Start of TaskManager (NIS Manager)		starts NIS Manager	2.5.14		1.8	v2.21.0	Y		Y
PRO-0006	2019-04-09	OBC_set_current_time		sets current time as OBC time	2.5.14	v2	1.8	v2.21.0	Y		
PRO-0006	2019-04-12	config_TC_TUGEGSE1_and_CLTU		aborts all waiting commands, starts data proxy, resets UV Status, sets TC SPACON Service mode to PACKET, changes Data Proxy profile to specified mode, waits for TC Link, sets TC SPACON to specified mode.	2.5.14		1.8	v2.21.0	Y		
MAT-0026	2019-04-19	with directives release_time := TIME end with;		releases a command at a certain time	2.5.14		2	v2.21.0		Y	
MAT-0027	2019-04-26	with directives release_time := TIME end with;		releases a command at a certain time	2.5.15		2	v2.21.0		Y	
PRO-0007	2019-04-29	R_DHS_N210		Enable live TM and set downlink channel	2.5.15	v4	2	v2.21.0	Y		Y
PRO-0008	2019-04-29	R_DHS_N215		Disable live TM downlink	2.5.15	v3	2	v2.21.0	Y		Y
MAT-0028	2019-04-29	dynamic_ptv := "overridden"		Sends command regardless of the TM link state	2.5.15		2	v2.21.0		Y	
PRO-0009	2019-04-29	config_DATA_PROXY_to_IDLE_mode		sets data proxy into IDLE mode	2.5.15		2	v2.21.0	Y		
PRO-0010	2019-04-29	config_TC_EGSE1_and_CLTU		sets data proxy into EGSE mode and TC spacon to CLTU mode	2.5.15		2	v2.21.0			Y
PRO-0011	2019-04-29	config_TC_to_OSUBBE_AND_PACKET		sets data proxy into OSUBBE mode and TC spacon to PACKET mode	2.5.15		2	v2.21.0			Y

Figure 11: Detailed pass events for phase "Configure (Close) Ground-Segment" (4a).

ID	DATE	NOTES	SOLVED
SCOS-0003	2019-04-05	The applications that should be stopped has to exist in SCTLconfig.xml If configured correctly can be seen when the EUD-SMH is opened and SCTL View is active. (full path will be displayed)	
PRO-0004	2019-04-05	see SCOS-0002	
PRO-0005	2019-04-05	see SCOS-0003	
CMD-0008	2019-04-09	current bug in data proxy: when set to idle 2 times in a row, the data proxy goes into an infinite loop out of which it cannot exit anymore. The data proxy has to restart to work properly again	
CMD-0009	2019-04-09	no problems found yet	
MAT-0022	2019-04-09	The peripheral that should be started has to be defined in SCTLconfig.xml to be called/started	
MAT-0023	2019-04-09	The peripheral that should be started has to be defined in SCTLconfig.xml to be called/started	
PRO-0006	2019-04-09	no problems found yet	
PRO-0006	2019-04-12	no problems found yet	
MAT-0026	2019-04-19	commands with release_time are aborted and show an error in log. (why the error occurs is not in the message)	see MAT-0027
MAT-0027	2019-04-26	Same issue as MAT-0026: commands with release_time are aborted and show an error in log. (why the error occurs is not in the message)	solved by correct configuration
PRO-0007	2019-04-29	with directive "dynamic_ptv := overridden" is not working -> therefore will only work when TM flow exists. WARNING: issue MAT-0028 has to be solved before this can be used in automation	
PRO-0008	2019-04-29	with directive "dynamic_ptv := overridden" is not working -> therefore will only work when TM flow exists. WARNING: issue MAT-0028 has to be solved before this can be used in automation	
MAT-0028	2019-04-29	Not working; Has to be solved by a patch from MATIS support. Issue raised in JIRA: MATIS-242	
PRO-0009	2019-04-29	no problems found yet	
PRO-0010	2019-04-29	procedure working, but if DATA PROXY is in idle mode and the drop down field shows the same mode as the procedure wants to change to, then the data proxy rejects the command and stays de-activated (idle mode)	
PRO-0011	2019-04-29	procedure working, but if DATA PROXY is in idle mode and the drop down field shows the same mode as the procedure wants to change to, then the data proxy rejects the command and stays de-activated (idle mode)	

Figure 12: Detailed pass events for phase "Configure (Close) Ground-Segment" (4b).

ID	DATE	ACTIVITY/PROCEDURE/SCHEDULE ID	COMMAND ID*	DESCRIPTION	VERSIONS							
					MATIS	PRO	MIB	OBSW	SUCCESS	FAILURE	WARNING
PRO-0012	2019-04-29	Abort_SMF_manual_stack		aborts all waiting commands, starts data proxy, resets UV Status, sets TC SPACON Service mode to PACKET, changes Data Proxy profile to specified mode, waits for TC Link, sets TC SPACON to specified mode.	2.5.15	v2	2	v2.21.0	Y		
PRO-0013	2019-04-29	OBC_set_current_time		sets onboard time to current time -1s	2.5.15	v2	2	v2.21.0	Y		Y
PRO-0014	2019-04-29	activate_live_TM_SBAND		activates live TM via SBAND	2.5.15	v2	2	v2.21.0	Y		Y
PRO-0015	2019-04-29	activate_live_TM_UHF		activates live TM via UHF	2.5.15	v2	2	v2.21.0	Y		Y
PRO-0016	2019-04-30	R_OBC_N525		set onboard time	2.5.15	v6	2	v2.21.0	Y		
PRO-0017	2019-04-30	R_OBC_N527		shift onboard time	2.5.15	v4	2	v2.21.0		Y	
MAT-0029	2019-04-30	relative_time		parameter type for relative time	2.5.15		2	v2.21.0		Y	
PRO-0018	2019-05-02	R_OBC_N425		Disable GPS time synchronisation	2.5.15	v4	2	v2.21.0	Y		
PRO-0019	2019-05-02	R_ADC_N230		Activate cADCS NADIRPOINTING mode	2.5.15	v10	2	v2.21.0	Y		Y
MAT-0030	2019-05-02	get TM parameter		gets TM parameter for TM CHECK	2.5.15		2	v2.21.0		Y	
PRO-0020	2019-05-03	R_GPS_N210		Power on GPS receiver	2.5.15	v8	2	v2.21.0	Y		Y
PRO-0021	2019-05-03	R_GPS_N220		Power off GPS receiver	2.5.15	v6	2	v2.21.0	Y		
PRO-0022	2019-05-03	R_ADC_N240		Activate cADCS EXPERIMENTAL mode	2.5.15	v9	2	v2.21.0	Y		
PRO-0023	2019-05-03	activate_AGGRETAION_AD01		Sets and activates Aggregation AD01	2.5.15	v3	2	v2.21.0	Y		
PRO-0024	2019-05-03	activate_live_TM_SBAND		activates live TM via SBAND	2.5.15	v3	2	v2.21.0	Y		Y
PRO-0025	2019-05-03	activate_live_TM_UHF		activates live TM via UHF	2.5.15	v2	2	v2.21.0	Y		Y
PRO-0026	2019-05-03	config_DATA_PROXY_to_IDLE_mode		configures data proxy to idle mode	2.5.15	v2	2	v2.21.0	Y		
MAT-0031	2019-05-08	with directives execution_time := XXXX end directives		sets the execution time of a command on board of the spacecraft	2.5.15	v2	2	v2.21.0		Y	
MAT-0032	2019-05-09	with directives authorisation := "XXX" end directives		decides whether command is loaded or loaded and sent	2.5.15	v2	2	v2.21.0	Y		
MAT-0033	2019-05-09	with directives execution_verification := "XXX" end directives		configures if command verification only for stages for ground or also if received on spacecraft	2.5.15	v2	2	v2.21.0	Y		

Figure 13: Detailed pass events for phase "Configure (Close) Ground-Segment" (5a).

	ID	DATE	NOTES		SOLVED
	PRO-0012	2019-04-29	no problems found yet		
	PRO-0013	2019-04-29	old version with "with directives release_time := .." not working -> workaroud of always setting the current time to current_time-1s and sending it directly		
	PRO-0014	2019-04-29	works if TM flow exists		
	PRO-0015	2019-04-29	works if TM flow exists		
	PRO-0016	2019-04-30	no problems found yet		
	PRO-0017	2019-04-30	procedure working, but DELTA time cannot be negative (see. MAT-0029) WORKAROUND: Use set time to a time in the past, then check again and shift to the correct time Issue raised in JIRA: MATIS-247		
	MAT-0029	2019-04-30	negative values are accepted on MATIS Server but cannot be entered in MATIS Designer or Operation Manager Issue raised in JIRA: MATIS-247		
	PRO-0018	2019-05-02	no problems found yet		
	PRO-0019	2019-05-02	problem with MATIS "Starting TM Live Provision" Application Unit not available		
	MAT-0030	2019-05-02	ERROR "Starting TM Live Provision; Application Unit not available" Possible Solution: reconfigure connection between MATIS and SMF Live provision service (needs further investigation before raising SPR)		
	PRO-0020	2019-05-03	OBSW: no confirmation in SCOS when gps receiver is powered on always gives FAILURE in confirmation stage (text even that it was turned on received and corresponding EPS was turned on)		
	PRO-0021	2019-05-03	no problems found yet		
	PRO-0022	2019-05-03	no problems found yet		
	PRO-0023	2019-05-03			
	PRO-0024	2019-05-03	problem see PRO-0007		
	PRO-0025	2019-05-03	problem see PRO-0007		
	PRO-0026	2019-05-03	no problems found yet		
	MAT-0031	2019-05-08	Command injection error (needs further investigation before raising SPR)		
	MAT-0032	2019-05-09	no problems found yet		
	MAT-0033	2019-05-09	no problems found yet		

Figure 14: Detailed pass events for phase "Configure (Close) Ground-Segment" (5b).

ID	DATE	ACTIVITY/PROCEDURE/SCHEDULE ID	COMMAND ID*	DESCRIPTION	VERSIONS				SUCCESS	FAILURE	WARNING
					MATIS	PRO	MIB	OBSW			
MAT-0034	2019-05-09	with directives release_time := XXXX end directives		sets the release time	2.5.15	v2	2	v2.21.0	Y		
MAT-0035	2019-05-10	configure MATIS for operational mode		closes connection to SVN and only uses exported Tag with validated procedures and schedules. Enables In-Tray functionality	2.5.15	v2	2	v2.21.0	Y		
MAT-0036	2019-05-10	In-Tray of Time-Scheduled MAUS		Automatic loading of MAUS when put into In-Tray	2.5.15	v2	2	v2.21.0	Y		
MAT-0037	2019-05-11	In-Tray of MAES		Automatic loading of MAES when put into In-Tray	2.5.15	v2	2	v2.21.0	Y		Y
MAT-0038	2019-05-10	In-Tray of Event-Scheduled MAUS		Automatic loading of MAUS when put into In-Tray	2.5.15	v2	2	v2.21.0	Y		
MAT-0039	2019-05-20	Create of LocalFile of ...		Creates a file in a certain directory on the local machine where SMF is running	2.5.15	v2	2	v2.21.0	Y		
MAT-0040	2019-05-20	WriteContent of LocalFile of ...		Writes content into a file	2.5.15	v2	2	v2.21.0	Y		
MAT-0041	2019-05-20	Delete of LocalFile of ...		Deletes a file in the local directory where SMF is running	2.5.15	v2	2	v2.21.0		Y	
PRO-0027	2019-05-21	send_command_get_return of SpaceShell		creates a file in which the executable command is written to and executes it on the space shell	2.5.15	v2	2	v2.21.0	Y		Y
PRO-0028	2019-05-21	execute_and_get_return of Command_Line		creates a file in which the executable command is written to and executes it in Command line	2.5.15	v2	2	v2.21.0	Y		Y

Figure 15: Detailed pass events for phase "Configure (Close) Ground-Segment" (6a).

ID	DATE	NOTES	SOLVED
MAT-0034	2019-05-09	no problems found yet	
MAT-0035	2019-05-10	no problems found yet	
MAT-0036	2019-05-10	no problems found yet	
MAT-0037	2019-05-11	Only absolute time set in MAES .xml file is used, no relative time possible (as used to in Operation Manager)	
MAT-0038	2019-05-10	no problems found yet	
MAT-0039	2019-05-20	no problems found yet	
MAT-0040	2019-05-20	care about special characters used by PLUTO -> use \ to	
MAT-0041	2019-05-20	Does not delete the file (but not necessarily needed for OPS-SAT)	
PRO-0027	2019-05-21	Does not delete the file (but not necessarily needed for OPS-SAT) (see MAT-0041)	
PRO-0028	2019-05-21	Does not delete the file (but not necessarily needed for OPS-SAT) (see MAT-0041)	

Figure 16: Detailed pass events for phase "Configure (Close) Ground-Segment" (6b).

E Excel Procedures to PLUTO Code Converter

This script converts a Excel procedure with a Standard defined by OPS-SAT into the PLUTO language for easier integration to MATIS.
The most recent code can be found in the following Github repository:
https://github.com/felixhessinger/opssat_converters/blob/master/ProcedureConverter_xlsx2pluto.py

F Excel Procedures to MATIS System ElementConfiguration File Converter

This script generates se.xml files needed for MATIS. In MATIS each System Element needs a se.xml file if minimum one procedure is included. The se.xml file contains information about the content of the System Element. These contents include procedure names, their configuration and their input arguments with description and ID.
The most recent code can be found in the following Github repository:
https://github.com/felixhessinger/opssat_converters/blob/master/SE_structureConverter_xlsx2seXml.py

G SCOS to MATIS MISC MIB Converter Python Code

This code converts MISCcontext.dyn files from SCOS to MISCconfig.dat files. MISCconfig.dat files are used by MATIS to read parameters from SCOS in MATIS and to react accordingly.
The most recent code can be found in the following Github repository:
https://github.com/felixhessinger/opssat_converters/blob/master/MATIS_MIB_MISC_dyn2dat_converter.py